A methodology for evaluating the capability of the Bradley 25mm Cannon to engage and defeat Pioneer class unmanned aerial vehicles

Wiley, Danny A.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A METHODOLOGY FOR EVALUATING THE CAPABILITY OF THE BRADLEY 25mm CANNON TO ENGAGE AND DEFEAT PIONEER CLASS UNMANNED AERIAL VEHICLES.

by
Danny A. Wiley

June, 1996

Thesis Advisor:	Robert B. McGhee
Second Reader:	Leroy A. Jackson

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE A METHODOLOGY FOR EVALUATING THE CAPABILITY OF THE BRADLEY 25mm CANNON TO ENGAGE AND DEFEAT PIONEER CLASS UNMANNED AERIAL VEHICLES.	5. FUNDING NUMBERS
6. AUTHOR(S) Wiley, Danny A.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Unmanned Aerial Vehicles (UAVs) represent a serious threat to forward deployed forces of the United States Army. The defense against such threats is currently provided primarily by the Bradley Stinger Fighting Vehicle (BSFV). The problem addressed is how to evaluate the effectiveness of the BSFV against a UAV. This thesis develops a computer simulation methodology for modeling the capability of a gun system to engage a UAV. Specifically, a review is made of the BSFV, BSFV 25mm Ammunition, and UAVs. These reviews formed the basis for a computer simulation, coded in Common Lisp Object System (CLOS), modeling the characteristics of three objects: a Projectile, a Launcher and a UAV. Although assumptions were made to simplify the model, simulation runs demonstrated that the rate of fire and aiming system used for launching projectiles resulted in one or more hits in 125 out of 154 engagement sequences. These engagement sequences were against a UAV flying at constant speed and altitude in crossing and inbound/outbound flight profiles. While all data used in this simulation were unclassified, the methodology presented could be used for further classified study, potentially producing a lower cost means for determining the effectiveness of air defense weapons against UAV threats.

14. SUBJECT TERMS Unmanned Aerial Vehicles, Bradley Stinger Fighting Vehicle, LISP Simulations, Air Defense	15. NUMBER OF PAGES 109
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

A METHODOLOGY FOR EVALUATING THE CAPABILITY OF THE BRADLEY 25mm CANNON TO ENGAGE AND DEFEAT PIONEER CLASS UNMANNED AERIAL VEHICLES.

Danny A. Wiley
Captain, United States Army
B.S., Appalachian State University, 1987

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1996

ABSTRACT

Unmanned Aerial Vehicles (UAVs) represent a serious threat to forward deployed forces of the United States Army. The defense against such threats is currently provided primarily by the Bradley Stinger Fighting Vehicle (BSFV). The problem addressed is how to evaluate the effectiveness of the BSFV against a UAV. This thesis develops a computer simulation methodology for modeling the capability of a gun system to engage a UAV. Specifically, a review is made of the BSFV, BSFV 25mm Ammunition, and UAVs. These reviews formed the basis for a computer simulation, coded in Common Lisp Object System (CLOS), modeling the characteristics of three objects: a Projectile, a Launcher and a UAV. Although assumptions were made to simplify the model, simulation runs demonstrated that the rate of fire and aiming system used for launching projectiles resulted in one or more hits in 125 out of 154 engagement sequences. These engagement sequences were against a UAV flying at constant speed and altitude in crossing and inbound/outbound flight profiles. While all data used in this simulation were unclassified, the methodology presented could be used for further classified study, potentially producing a lower cost means for determining the effectiveness of air defense weapons against UAV threats.

TABLE OF CONTENTS

I. INTRODUCTION 1
  A. GOALS 1
  B. BACKGROUND AND MOTIVATION 1
  C. RESEARCH QUESTIONS 2
  D. ORGANIZATION 3
II. UNMANNED AERIAL VEHICLES 5
  A. BACKGROUND 5
  B. GENERAL OVERVIEW 6
    1. Capabilities and Value 6
    2. Operation 6
  C. PIONEER UNMANNED AERIAL VEHICLE 7
    1. Description 7
    2. Operational Status 8
    3. Specifications 8
  D. CURRENT UAV PROGRAMS 9
III. BRADLEY FIGHTING VEHICLE 11
  A. GENERAL OVERVIEW 11
  B. BRADLEY STINGER FIGHTING VEHICLE 12
    1. Introduction 12
    2. Description 12
    3. Operational Status 13
    4. Characteristics of the BSFV 13
    5. 25mm Cannon 14
    6. Ammunition Assessment 16
      (a) Introduction 16
      (b) M792 High-Explosive 17
      (c) M791 Armor-Piercing 17
IV. SIMULATION MODEL 19
  A. INTRODUCTION 19
  B. GENERAL OVERVIEW 19
  C. SCENARIO 19
  D. CHARACTERISTICS OF UNMANNED AERIAL VEHICLE 20
  E. CHARACTERISTICS OF PROJECTILE AND THE LAUNCHER 22
  F. DETERMINING A HIT 23
  G. CHARACTERISTICS OF THE PERFECT GUNNER 27
    1. Adjust Fire Elevation 27
      a. 0 Degree Offset Adjust Fire Elevation 27
      b. 45 Degree Offset Adjust Fire Elevation 28
    2. Adjust Fire Azimuth 29
  H. FLIGHT PROFILES ANALYZED 30
  I. LIMITATIONS AND ASSUMPTIONS 31
  J. SUMMARY 32
V. RESULTS 33
  A. 0 DEGREE OFFSET INBOUND/OUTBOUND 33
    1. Introduction 33
    2. Analysis 35
  B. 45 DEGREE OFFSET CROSSING 35

1. Introduction 35
2. Analysis 38

VI. CONCLUSIONS AND RECOMMENDATIONS 39

A. CONCLUSIONS 39

B. RECOMMENDATIONS FOR FUTURE RESEARCH 39

APPENDIX A: SCREEN IMAGES 41

APPENDIX B: SOURCE CODE (UAV AND PROJECTILE FILES) 45

APPENDIX C: SOURCE CODE (STROBE CAMERA FILE) 55

APPENDIX D: SOURCE CODE (PERFECT AUTOPILOT FILE) 63

APPENDIX E: SOURCE CODE (DETERMINING A HIT FILE) 69

APPENDIX F: SOURCE CODE (ADJUST FIRE FILE) 73

APPENDIX G: SOURCE CODE (MOUSE HANDLER FILE) 83

APPENDIX H: SOURCE CODE (ROBOT KINEMATICS FILE) 87

APPENDIX I: SOURCE CODE(COMPILE AND LOAD FILES) 93

LIST OF REFERENCES 95

INITIAL DISTRIBUTION LIST 97

ACKNOWLEDGEMENTS

This research was possible due to the efforts of many people. Most notably is that of my thesis advisor and second reader. The individual attention, patience and enthusiasm displayed by my thesis advisor, Professor Robert B. McGhee, allowed me to accomplish things I never thought possible. Major Leroy A. Jackson, my second reader, provided the coaching I needed to put ideas and concepts on paper as well as keeping me honest with the analysis and conclusions in the thesis.

Sincere thanks go out to two fellow students. Mr. Duane Marhefka provided the formulas and concepts in the hit algorithm in Chapter IV. LT Duane T. Davis provided his expertise in programming ideas and teaching technical skills used throughout this publication. Both of these individuals dedicated many hours of teaching and assistance. Their selfless efforts to help me achieve many of my research goals are a tribute to the spirit of academics.

I would also like to thank the students and faculty members at the Naval Postgraduate School (NPS), I have found the educational experience to be a positive one and this is only possible with the outstanding faculty and peers with which this institution is blessed.

Appreciation goes out to my good friend LCDR Timothy J. Werre who provided a listening ear always and a respected opinion when necessary.

Finally, I am grateful to my wife Gabriele and daughter Jennifer for their patience, understanding, and support throughout this research and my studies.

I. INTRODUCTION

A. GOALS

The goal of this thesis is to develop a method to determine the ability of the Bradley Stinger Fighting Vehicle (BSFV) 25mm Cannon to counter the emerging threat of Unmanned Aerial Vehicles (UAVs). The intent is to simulate the BSFV engagement of a UAV, to analyze the results to understand this capability, and to provide insight for improving existing weapons, munitions and systems. This work develops an approach and is unclassified. Bradley effectiveness results contained herein do not represent actual operational characteristics, but are illustrative of an application of the simulation system described.

B. BACKGROUND AND MOTIVATION

As the Army evolves toward the Twenty First Century, the need for knowledge about the capability of its weapons against emerging threats becomes more important. Commanders must improve their ability to detect, locate, identify and engage targets at maximum range. UAVs will likely be a target on the battlefields of the Twenty First Century. Research today may provide commanders at all echelons with an awareness of the capabilities possessed by today's weapons and needed by tomorrow's weapons to effectively engage and defeat a hostile UAV.

A UAV can perform a wide variety of missions, but such vehicles are primarily used to gather information about a specific area of interest. As capabilities increase, the mission of the UAV will be similar to that of manned aerial vehicles. The relatively low cost of such unmanned vehicles, along with the ability to accomplish a mission without risking human life, are a valuable asset for military forces around the world. The increased use of the UAV by other countries leads to concerns about actions to counter this threat.

Defending against a UAV is the shared responsibility of several components on the battlefield. One such component is the Forward Area Air Defense (FAAD) weapons of an Army Division. The FAAD Weapons that engage an air target are: [Ref. 19]

- ***Line-of-Sight-Rear (LOS-R) FAAD Weapon.*** A system composed of a missile component and a machine gun that is Identification Friend or Foe (IFF) capable. The Army's current system is the AVENGER. The AVENGER has an organic Forward Looking Infra-Red (FLIR) sensor.

- ***Man Portable Air Defense System (MANPADS).*** A shoulder fired system with an IFF capable missile. The Army's current system is the Stinger. It has no organic sensor capability.

- ***Bradley Fighting Vehicle (BSFV).*** A system composed of a Bradley Fighting Vehicle (BFV) with an externally mounted Stinger weapon system. The BSFV has a 25mm cannon and Stinger missiles on-board with no organic sensor capability.

It is imperative to study the capability of these weapons to defend against a UAV. A FAAD weapon system is very likely to encounter a UAV flying at low altitude that is difficult to detect and engage. The potential widespread use of UAVs on the battlefields of tomorrow requires the study of the effectiveness of today's weapons against this threat. Such research will help planners design weapons to counter the future UAV threat. This thesis will focus on the capability of the BSFV 25mm Cannon to engage and hit a UAV.

C. RESEARCH QUESTIONS

This thesis will examine the following research areas:

- Review the physical capabilities to bring fire to bear on a target such as the cannon elevation limits and the operational flight characteristics of a UAV.
- Analyze the types of ammunition available.
- Determine the potential of a hit based on:
    - the simulation of a projectile, constrained by the characteristics of the Bradley Cannon and ballistics data derived from unclassified specifications.

- the simulation of a UAV, derived from the unclassified specifications and operational characteristics of a Pioneer UAV.

D. ORGANIZATION

Chapter II of this thesis provides a general overview of unmanned aerial vehicles and provides a detailed description of the Pioneer UAV. Chapter III provides a detailed description of the Bradley Fighting Vehicle and its role as an air defense weapon. Chapter III looks specifically at the 25mm Cannon and its ammunition. Chapter IV outlines the UAV and Cannon model and provides the limitations, assumptions and profiles for the simulation. Chapter V details the results of the simulations and analyzes the data and provides an evaluation of the results. Chapter VI presents conclusions and recommendations for further research. The source code is provided in the Appendices A - I.

II. UNMANNED AERIAL VEHICLES

A. BACKGROUND

Although UAVs were envisioned as far back as 1916, the first modern UAV was an unmanned B-17 controlled from a second aircraft. The controlling aircraft accompanied the unmanned B-17 to a target area where it was then remotely flown closer to the actual target. These unmanned B-17 aircraft were used extensively during the atomic bomb tests in the South Pacific to monitor radiation. [Ref. 13]

Following the Soviet interception of Francis Gary Powers' U-2 in 1960, the Air Force and other national agencies directed resources into UAV development programs. The AQM-34 was one such system. Variations of the AQM-34 flew more than 17,500 missions around the world from 1958 to 1975. [Ref. 13]

A highly successful system, known as Buffalo Hunter, flew more than 1,600 missions in Southeast Asia. It was a variation of the AQM-34 that usually operated at altitudes of more than 60,000 feet. The operational concept later evolved to include very low-level photo-reconnaissance missions over North Vietnam. Other missions included signal intelligence (SIGINT) and psychological operations missions flown by the 100th Strategic Reconnaissance Wing. The last mission of Buffalo Hunter took place over Saigon on April 30, 1975 during the final stages of the United States' evacuation. [Ref. 13]

Post-Vietnam era UAV developments were led primarily by the Central Intelligence Agency and the U.S. Army. The Field Artillery's Aquila target acquisition, designation and reconnaissance system (TADARS) began development in 1974, but was terminated in 1987 after extensive testing. In 1985 the U.S. Army Intelligence Center at Fort Huachuca, Arizona was designated as the UAV proponent, less Aquila, to develop a family of UAVs to provide organic, near real-time support to battlefield commanders. The requirements for deep and close UAVs were approved by the Training and Doctrine Command (TRADOC) in December 1988. [Ref. 13]

B. GENERAL OVERVIEW

1. Capabilities and Value

Since the Vietnam conflict, the ability of UAVs to gather and return sensitive intelligence has improved as the UAV has grown smaller, lighter and quieter with increased operational range and flight endurance. [Ref. 7] Current UAV systems can provide real-time multi-spectral video to ground forces in less than a minute while the aircraft is being remotely piloted from either a permanent control station at sea or a portable station on the ground. The high resolution video platforms, which usually have day and night image capability, are gyro-stabilized. On average, today's UAVs can fly for up to six hours and have ranges of 100-300 miles from their control station. These aircraft can often fly further if control is transferred to a forward station. [Ref. 12]

The advantage these UAVs have over other aircraft with similar imaging capabilities is their relative low cost without risk of human life. Even though NATO peacekeepers have met little resistance (as of this publication) in Bosnia, bad weather can still pose a threat to manned aircraft [Ref. 18]. UAVs can also be used in advance of other air units to check terrain for possible unmarked hazards like low power lines.

2. Operation

An *internal* pilot flies the UAV from the control station using gauges and on-board cameras; *external* pilots fly the aircraft by sight and are directly responsible for control of the plane's take-off and landing procedures. Once a mission is recorded, the plane is launched and taken to a set altitude and heading by the external pilot. The mission is then handed off to the internal pilot who controls the duration of the mission until return for recovery or landing. Most UAVs are outfitted with an autopilot system, navigational equipment, and communications capabilities that allow the plane to be flown in an autonomous mode with human monitoring. Once missions are underway, the person at the control station is able to modify parameters for future legs of the flight plan. Weather conditions, for example, often require a change in altitude to allow the point of interest to be viewed. [Ref. 12]

An example of the operation of a UAV for a military style deep fire mission would involve the use of two UAVs with ranges of 200 kilometers each. One UAV flies a mission to locate and confirm a suspected missile launch site. The other UAV is used as an airborne communications relay station, facilitating the information flow from the UAV on the missile site to the planners in the rear area of the battlefield. This simple example illustrates the operation of the UAV in two roles: as a reconnaissance vehicle and as a communications relay station.

C. PIONEER UNMANNED AERIAL VEHICLE

1. Description

The pioneer aircraft has a shoulder-wing, pod-and-twin-tailboom configuration, with fixed tricycle landing gear and an arrester hook. Payloads are mounted in a small slot located on the front belly of the aircraft. The wings, booms and tail unit are detachable. This simplifies dismantling and assembly in the field and facilitates the prompt dispatch of several vehicles at one time. [Ref. 7]

A Pioneer fitted with standard electronic equipment for a reconnaissance mission is depicted below.

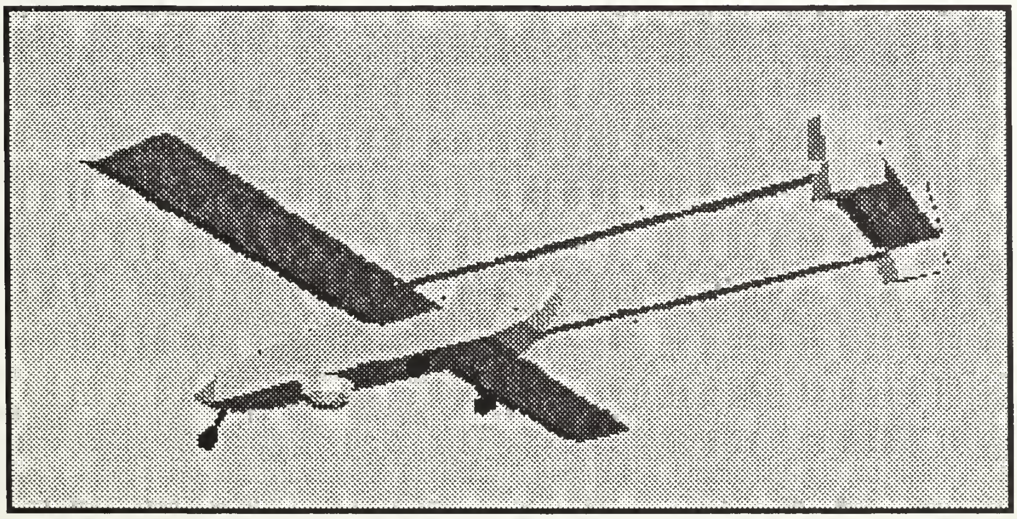


Figure 1: Computer Generated Pioneer UAV Image

The Pioneer operates at ranges up to one hundred nautical miles and altitudes up to twelve thousand feet for as long as five hours. Pioneer carries an independently controlled high resolution daytime TV or a nighttime infrared camera which transmits real-time imagery back to a ground or ship-based control station. The Pioneer can take off from a fixed airport runway on its own power or it can be launched from the ship's flight deck with a rocket assist. Similarly, the Pioneer lands at land based sites with runways or it is recovered by a fifty foot high net raised over a ship's flight deck. The Pioneer cruises at between sixty and one hundred knots.

Continuous communications between the ground control station and the UAV allow for constant control and monitoring of Pioneer's flight systems, navigation, and video payload. A single operator can fly and control a Pioneer. Once the craft is airborne, it is controlled by the control station. This station is equipped with high resolution color graphics displays. The control station operator can manipulate the cameras and other equipment on board the Pioneer. All imagery data can be frozen, stored, and recorded for future processing. [Ref. 7]

The control station supports graphical entry of a mission flight plan on a digitized color map display. During flight operations, the map display shows the UAV's position and the payload's optical axis. The control station can automatically transmit commands to the UAV to fly each leg of the mission flight plan. Another option is to transmit the entire flight plan to the UAV, which will then fly the plan autonomously. Control stations can be deployed in a manner to increase the operational range and flexibility of the Pioneer. [Ref. 7]

2. Operational Status

The Pioneer was used successfully in the Gulf War and continues to be used by the U.S. Navy, U.S. Army, and U.S. Marine forces [Ref. 7]. The Pioneer is currently used for surveillance, over-the-horizon targeting, naval gunfire spotting, overland reconnaissance, and real-time battle damage assessment.

3. Specifications

A Pioneer UAV system is composed of five air vehicles, nine payloads, one ground control station, one pilot control station, one launcher, and a recovery system. Table 1 contains details of the air vehicle specifications.

AIR VEHICLE	
Power Plant	19.4 kW (26 hp) Sachs SF 350 2 cylinder 2 stroke engine
Wing Span	5.11 m
Length	4.26 m
Max Payload	45 kg
Max Speed	176 km/h
Cruising Speed	120 km/h
Loiter Speed	111 km/h
Max S/L Rate of Climb	4.1 m/s
Operating Height Range	305-3660 m
Data Link Range	185 km
Endurance	5.5 hrs

Table 1: Specifications of Pioneer UAV [Ref. 7]

D. CURRENT UAV PROGRAMS

Congressional interest has focused on developing UAVs that have a common architecture and can interact on the battlefield. To ensure this coordinated effort, in 1988 Congress halted all service UAV funding and established the Joint Program Office for UAVs (JPO-UAV). This office was chartered to develop a Department of Defense (DoD) master plan for UAVs and establish a family of common interoperable UAVs. [Ref. 13]

Since 1989, this family of UAVs has evolved into the UAV-Short Range (UAV-SR), UAV-Close Range (UAV-CR) and UAV-Endurance (UAV-E). Table 2 shows the operational characteristic of each type of UAV. [Ref. 13]

	UAV-CR	UAV-SR	UAV-E
Fielding Date	FY98	FY95	FY98
Range	50 km	200 km	500 nm
Flight Endurance	3 hrs	10 hrs	24 hrs
Service Ceiling	13,000 AGL	15,000 AGL	25,000 AGL

Table 2: Specification Classes of Current UAV Programs [Ref. 13]

III. BRADLEY FIGHTING VEHICLE

A. GENERAL OVERVIEW

The Bradley Fighting Vehicle was designed to be a common vehicle for both the infantry and scout roles. The Bradley Fighting Vehicle is made of all-welded aluminum armor and is about seven meters long, three meters wide, and three meters high. The Bradley has two main components: the chassis and the turret. [Ref. 15]

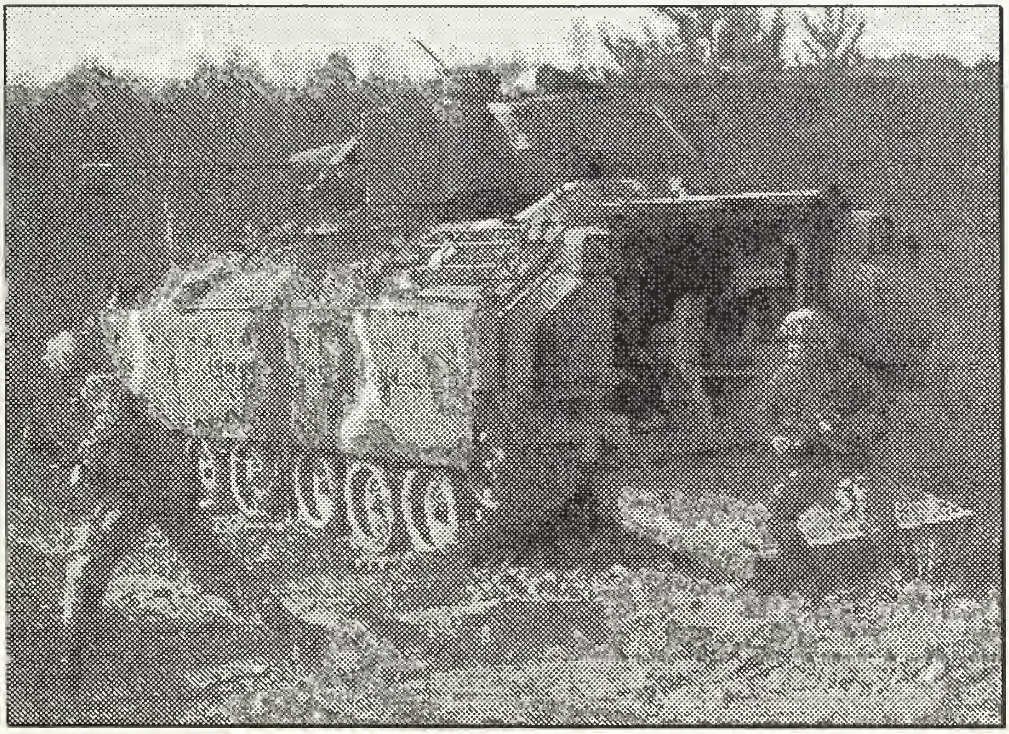


Figure 2: Bradley Fighting Vehicle

The chassis is powered by a Cummins VTA-903T diesel engine with a Martin Marietta HMPT-500 hydromechanical transmission. The turret contains the main armament consisting of a McDonnell Douglas M242 25mm Chain Gun. Additionally the turret contains a 7.62 mm M240C coaxial mounted machine gun to the right of the 25mm cannon. [Ref. 14]

Since its initial fielding in 1983, the Bradley has taken on many different configurations and roles. Some of these are: Personnel Carrier, Cavalry Fighting Vehicle, and Bradley Stinger Fighting Vehicle. Variants of the Bradley Fighting Vehicle are equipped with the Hughes TOW anti-tank missile and the Stinger anti-aircraft missile system. The chassis of this vehicle is also used for the Multiple Launch Rocket System (MLRS) and serves as an ammunition carrier. Currently contractors are building various Bradley based prototype systems to replace older systems mounted on the M113 chassis such as the Bradley Fire Support Team Vehicle.

B. BRADLEY STINGER FIGHTING VEHICLE

1. Introduction

Following the withdrawal of the M163 20mm self-propelled Vulcan Air Defense system, the U.S. Army began using the Bradley Fighting Vehicle to transport Stinger missiles and the soldiers that employ them. This combination of the Bradley Fighting Vehicle with the Stinger teams was the initial Bradley Stinger Fighting Vehicle (BSFV). [Ref. 14]

2. Description

The current BSFV has a firing pod containing four Stinger missiles on the left side of the turret (see Figure 3). The main armament for the system is the 25mm cannon and the Stinger missiles. This version of the Bradley has a five man crew consisting of a squad leader, a senior gunner, a driver and two Stinger gunners. The Stinger missiles can be fired from the pod or mounted on to grip stocks for dismounted shoulder firing. [Ref. 14]

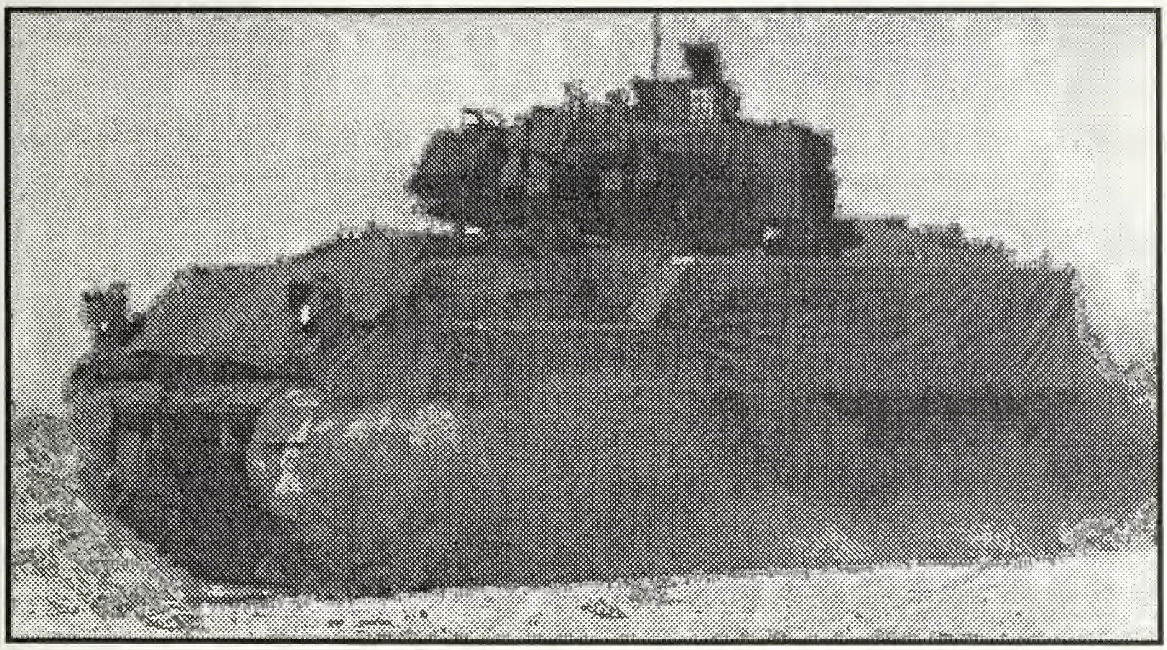


Figure 3: Bradley Stinger Fighting Vehicle

3. Operational Status

The BSFV is currently fielded to all U.S. Army Air Defense Battalions assigned to Mechanized Divisions. The BSFV is the most forward deployed Air Defense weapon on the battlefield and performs the short range (less than five kilometers) Air Defense mission for the U.S. Army. Although the BSFV is not a combat proven Air Defense weapon, the Bradley's 25mm cannon reportedly exceeded all expectations during the Persian Gulf War. During combat operations in the Persian Gulf, the Bradley maintained an operational readiness rate above ninety percent. The BSFV has performed extremely well in rigorous live fire tests and field testing at the National Training Center (NTC) at Fort Irwin, California. [Ref. 14]

4. Characteristics of the BSFV

The BSFV is a lethal weapons platform that enhances the firepower and survivability of air defense assets on the battlefield. Table 3 below outlines the characteristics of the BSFV.

[illegible]		
COMBAT WEIGHT	23.5 tons	
HEIGHT	116 inches	
WIDTH	128 inches	
GROUND CLEARANCE	18 inches	
ACCELERATION (0 TO 32 KMPH)	7.7 seconds	
ROAD SPEED	66 kmph	
CROSS-COUNTRY SPEED	48 kmph	
CRUISING RANGE	480 km	
VERTICAL OBSTACLE	36 inches	
TRENCH	100 inches	
FORDING DEPTH	3.5 feet	
MAIN ARMAMENT	25mm (900 rounds)	
	Tow (5 rounds)	
	Stinger (6 rounds)	
MAXIMUM EFFECTIVE RANGE	25mm ground	3000 meters
	25mm air	2000 meters
	Stinger	4000 meters+
	Tow	3700 meters
	Coax 7.62mm	900 meters

Table 3: General Characteristics of the BSFV [Ref. 15]

5. 25mm Cannon

[illegible]	
Elevation	+60 degree to -10 degree
Traverse	360 degree continuous
Sight	Thermal, Direct View Optics
Sight Magnification	3x and 12x
Firing Rate - Low	75-125 rounds per minute
Firing Rate - High	175-225 rounds per minute

Table 4: General characteristics of the BSFV Gun [Ref. 15]

The 25mm cannon subsystem is a weapon system designed to attack and defeat enemy armored vehicles and other targets, such as aircraft, using armor-piercing (AP) or high-explosive (HE) projectiles. The gun is an electrically powered, chain driven automatic weapon. It is fed by a metallic link and has dual feed capability. The 25mm ammunition cans hold up to seventy rounds of AP and two hundred and thirty rounds of HE ammunition. The 25mm Cannon has three parts: the barrel assembly, feeder assembly, and receiver assembly. The gun has both electrical and manual fire control. [Ref. 16]

The major components and functions of the 25mm cannon are:

- The Barrel Assembly
    - gives directional control to the projectile and
    - includes a muzzle brake that reduces blast flash and absorbs some of the recoil.
- The Feeder Assembly
    - removes linked ammunition from feed chutes,
    - strips rounds from links,
    - places round into bolt face,
    - removes spent cartridge case from bolt face,
    - provides a means of selecting two types of ammunition,
    - provides a means to manually operate the gun during power failure, and
    - sends electrical signals to the turret controls indicating position of the bolt.
- The Receiver Assembly
    - rams and fires the rounds,
    - extracts and ejects spent cartridge cases or unfired rounds,
    - suppresses the recoil force from barrel and breech, and
    - contains a mechanical interlock mechanism that stops the gun if a round misfires or hangfires. [Ref. 15]

The 25mm Cannon is normally operated in the electrical mode. The weapon control box allows for selection of the 25mm Cannon, ammunition type (AP or HE), rate of fire, and arming of the 25mm Cannon. Firing rate can be single shot, low rate (around 75 to 125 rounds per minute) or high rate (around 175 to 225 rounds per minute). Indicator lights on the weapon control box show which mode is selected. Weapon select will not function if the gun is out of sear. When AP or HE is selected, the feed select solenoid is enabled. Once the feed selection is made and the gun is armed, it can be fired using one of three triggers (gunner's, commander's hand station, or traverse hand wheel). When the trigger is pressed, the sear solenoid is energized releasing the sear pin from the master link on the chain. The sear pin action energizes the electric motor which drives the track and bolt assembly, and feeder. The feeder places a live round in front of the bolt and the bolt moves forward and locks to the breech. When the gun is in breech locked position, the sear pin is engaged against the chain's safety link. When the round is fired, the bolt unlocks and the sear pin is released allowing the cycle to continue. The spent case is extracted by the rearward motion of the bolt. The rotor then turns to place the spent case into the eject chute while a new round is placed in front of the bolt. In single shot and low rate, the motor is turned off and the sear pin is engaged to contact the master link and stop the gun in the sear position. On the next forward motion of the bolt, the spent case is pulled out of the ejection chute by the ejector on the bolt carrier. [Ref. 16]

Ammunition for the 25mm cannon is stored below the weapon in an ammunition can. The ammunition can has two sections for AP and HE ammunition. Each ammunition belt feeds through its own forwarder and chute. The forwarders are manually operated and used while loading ammunition. The ammunition can has two sensors that tell when the 25mm cannon is low on ammunition. This leaves the end of the ammunition belt in a position that allows the ammunition to be easily attached to the old belt. [Ref. 16]

6. Ammunition Assessment

a. Introduction

Two types of ammunition are used by the BSFV's 25mm cannon. The ammunition characteristics are outlined in Table 5 below.

AMMUNITION	CLASSIFICATION	IDENTIFICATION	FUZE
M791 APDS-T	ARMOR PIERCING	BLACK TIP	NONE
M792 HEI-T	HIGH EXPLOSIVE INCENDIARY TRACER	YELLOW	M758

Table 5: Two Types of Ammunition [Ref. 16]

Target identification is the critical element in ammunition selection. Both the M791 and M792 can be used against slow moving, fixed-wing aircraft and helicopters. Rounds should be fired in twenty to twenty-five round bursts with the high rate of fire selected. The M791 has a higher probability of hit than M792; however, the M792 has a higher probability of kill given a hit. At ranges beyond twelve hundred meters, the M791 is more effective against helicopters. At ranges less than twelve hundred meters, the M792 is more effective against helicopters. Beyond two thousand meters, the 25mm cannon loses its effectiveness to kill. [Ref. 16]

b. M792 High-Explosive

The M792 projectile is a high-explosive incendiary tracer that is commonly referred to as a "heat" round. Because the projectile is explosive in nature, if it hits the target it causes the more damage to an air target than the M791 Armor-Piercing projectile. Its characteristics are given in Table 6 below.

M792 High-Explosive	
Specified Muzzle Velocity	1100 meters per second
Drop at 300 meters	-0.402 meters
Drop at 500 meters	-1.200 meters
Drop at 700 meters	-2.542 meters
Velocity dispersion	+/- 5 meters per second
Angular dispersion	+/- .89 mils

Table 6: M792 Projectile [Ref. 17]

c. M791 Armor-Piercing

The M791 projectile is a armor piercing projectile that is commonly referred to as a "SABOT" round. It has a higher probability of hit against an air target than the M792 High-Explosive projectile. Its characteristics are given in Table 7 below.

M791 Armor-Piercing	
Muzzle Velocity	1345 meters per second
Drop at 300 meters	-0.251 meters
Drop at 500 meters	-0.708 meters
Drop at 700 meters	-1.413 meters

Table 7: M791 Projectile [Ref. 17]

IV. SIMULATION MODEL

A. INTRODUCTION

This chapter describes the UAV and projectile object models and the algorithms that simulate the gunner and determine when a projectile hits the UAV. The UAV flight profiles are introduced as are the object model limitations and assumptions. The models were implemented in Franz Common Lisp Object System (CLOS) using Allegro Common Windows for graphics display.

B. GENERAL OVERVIEW

This simulation model uses data from unclassified sources and personal insight from military experience and interviews with operators of the UAV and Bradley Fighting Vehicle systems. In general, for this simulation to provide accurate results, the author must address:

- the limitations and assumptions about the environment, and
- the assumptions about the UAV and Projectile behavior.

This model represents a methodology that provides insight into the potential effectiveness of a BSFV against an unmanned aerial craft.

C. SCENARIO

The general approach in this model is for the gun to fire at a sustained rate of fire at an aim point in front of the aircraft. In this method of firing, adjustments are determined by firing at the target and correcting the aim based on observations from a second individual (the squad leader). The gunner will attempt to establish a volume of fire in front of the aircraft so that the aircraft flies into the projectiles. After the aircraft has flown through the wall of projectiles, the gunner will adjust the aim point to once again establish a volume of fire in front of the aircraft. This continues until the aircraft is either destroyed or out of range. This scenario is modeled through the simulation of a UAV in flight and projectiles fired from a launcher described in the following sections.

D. CHARACTERISTICS OF UNMANNED AERIAL VEHICLE

The characteristics of the Pioneer UAV described in Chapter II and information obtained during conversations with experienced Pioneer UAV pilots are the basis of the UAV model in this simulation. A comparison of Tables 8 and 9 below with Table 1 shows that the simulated UAV is somewhat larger than the Pioneer UAV, although otherwise very similar, in order to represent the average size of UAVs of the same class as the Pioneer UAV.

The mission and flight profile of the UAV is comparable to that of a Pioneer UAV conducting reconnaissance against armored vehicles in the forward battle area. The modeled UAV flies at altitudes of 2500 or 3500 feet and at an operating speed of 65 knots. The altitude is selected to maintain line of sight between the aircraft and its controller. This line of sight restriction forces the aircraft to maintain a higher altitude at greater distances from the controller. The speed is a function of the UAV type, the payload, and the mission; in general, 65 knots is the operating speed selected for reconnaissance of the forward edge of the battlefield. [Ref. 18]

[illegible]	
Wing Span	24 feet
Length	29 feet
Operating Speed	74.80 mph or 65 knots (110 feet per second)
Operating Altitude	2500 or 3500 feet

Table 8: Characteristics of Simulated UAV

The simulated UAV is composed of thirteen rectangular surfaces. Each surface is independently tested for penetration by a projectile. The UAV surfaces and their dimensions are listed in Table 9 below.

Simulated UAV Surface	Width	Length
Body		
Belly and Top	2 feet	14 feet
Nose and Rear	3 feet	2 feet
Sides, Left and Right	3 feet	14 feet
Wings, Left and Right	3 feet	11 feet
Tail Extensions, Left and Right	1 feet	16 feet
Tail	3 feet	10 feet
Tail Fins, Left and Right	3 feet	3 feet

Table 9: Simulated UAV Surfaces

The line drawings in Figure 4 below illustrate two aspects of the simulated UAV. Any projectile that penetrates a surface is classified as “hitting” the UAV in that area.

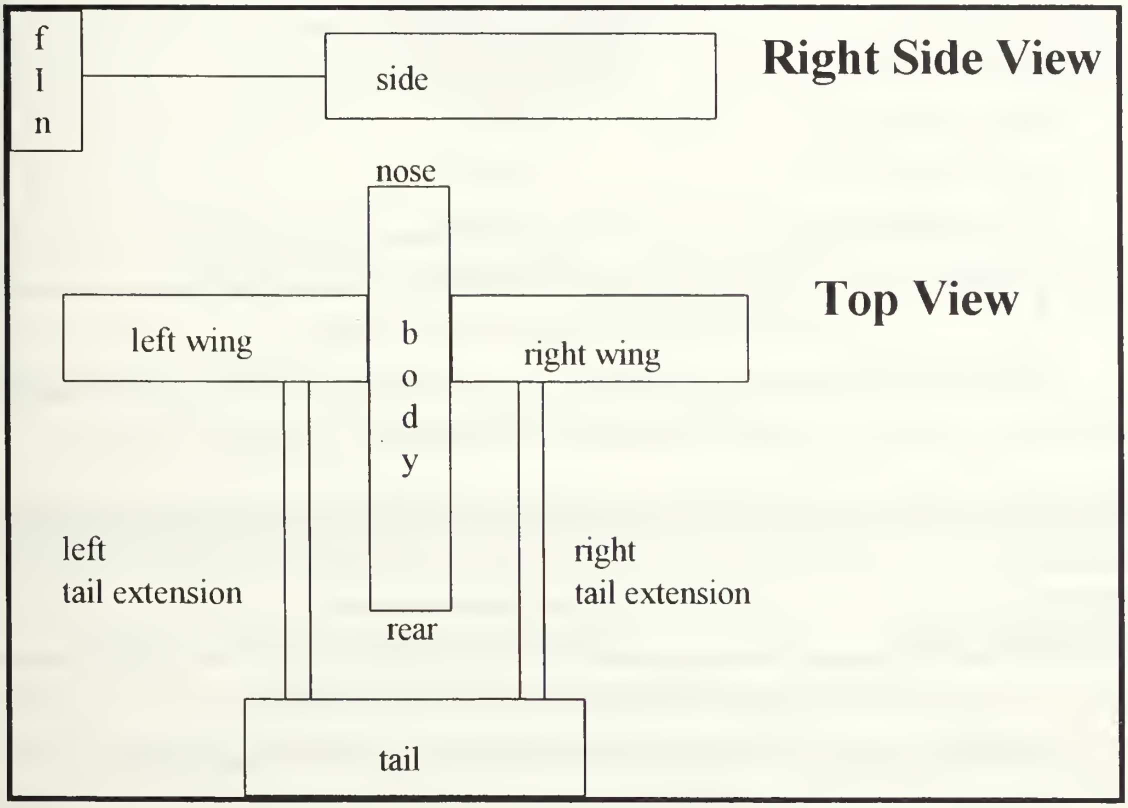


Figure 4: Line Drawing of UAV

Appendix A contains the Lisp Code that generates the graphical display and models the flight of the UAV.

E. CHARACTERISTICS OF PROJECTILE AND THE LAUNCHER

The projectile was modeled after the M792 High-Explosive 25mm round discussed in Chapter III. The muzzle velocity is 3608 feet per second and the drop characteristics of the round were derived using Heun integration with the data in Table 6. This method results in the drop values listed in Table 10 below. The dynamic model used to obtain these results assumes no aerodynamic lift acting on the projectile and drag proportional to the square of the velocity. In the absence of further information, and for simplicity, angular and velocity dispersion figures were modeled as uniformly and independently distributed random variables with zero mean and range equal to plus or minus the given dispersion figures.

[illegible]	
Velocity Dispersion	16.4 feet per second
Angular Dispersion	0.05006219 degrees
Average Muzzle Velocity	3608 feet per second
Drop at 1031 feet	1.44 feet
Drop at 1665 feet	4.02 feet
Drop at 2260 feet	7.89 feet
Effective Range	6100 feet

Table 10: Simulated Projectile Characteristics

The simulated launcher characteristics are listed in Table 11 below. The simulated launcher has a single rate of fire while the BSFV has both a low and high rate of fire.

[illegible]	
Elevation	+60 degree to -10 degree continuous
Traverse	360 degree continuous
Firing Rate - Burst	300 rounds per minute (1 every .2 seconds)

Table 11: Simulated Projectile Launcher Characteristics

Appendix A contains the Lisp code that models the projectile and launcher.

F. DETERMINING A HIT

The algorithm that determines if a projectile has penetrated any of the simulated UAVs surfaces has four steps. Step 1 screens out all projectiles that are too distant from the UAV. This step is illustrated in Figure 5 below.

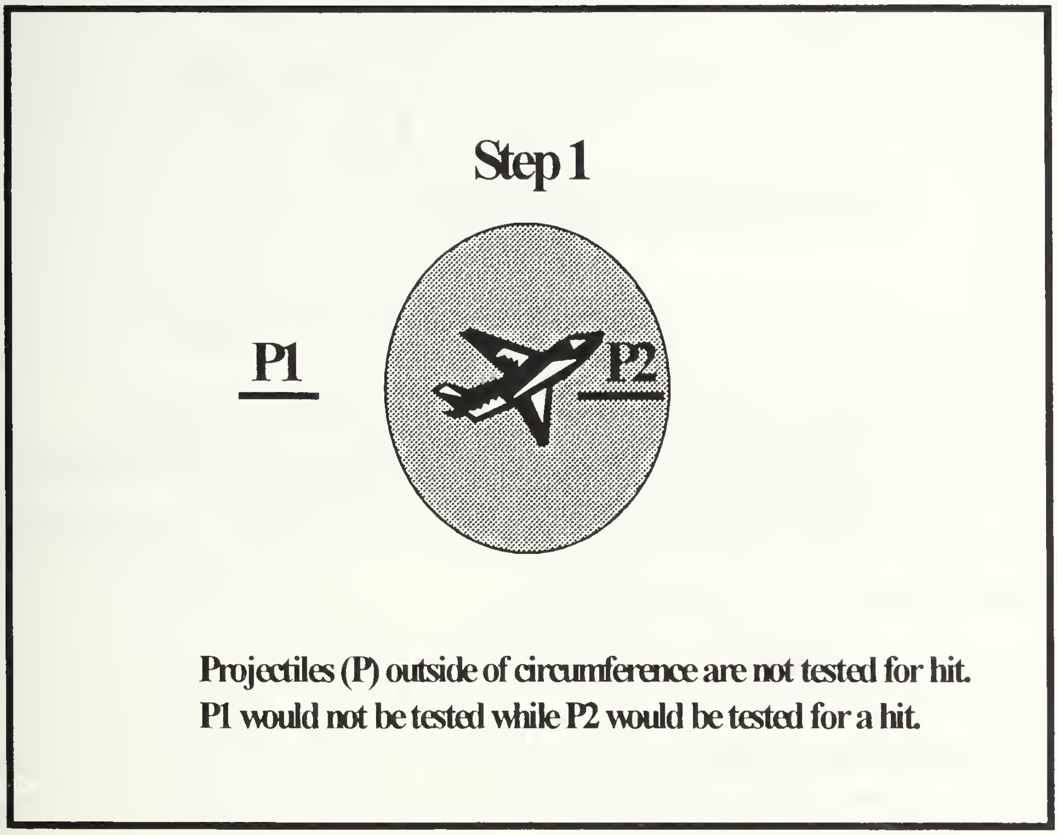
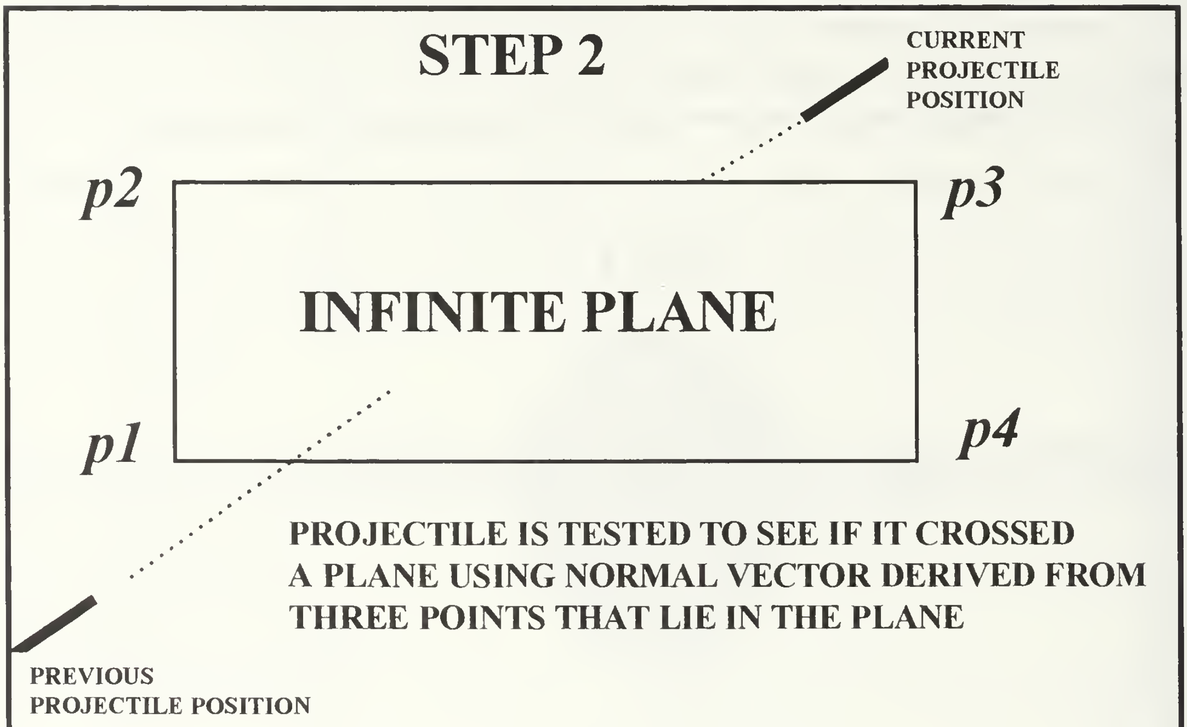

Step 1
P1
P2
Projectiles (P) outside of circumference are not tested for hit.
P1 would not be tested while P2 would be tested for a hit.

Figure 5: Step 1 of Hit Algorithm

Step 1 tests if the projectile is within 225 feet of the center point of the UAV. This distance is chosen because of the projectile velocity and the time step used in the simulation. The minimum range from the UAV to the projectile is 2500 feet (equal to the minimum UAV altitude). Analysis of the projectile at 2500 feet using Heun integration indicates that the projectile travels less than 225 feet in a time step. (This follows because a projectile fired at zero degrees elevation travels 223 feet in the next 1/10th second after reaching the 2500 foot minimum range for engagement.) Step 1 screens out computationally costly calculations for projectiles incapable of hitting the UAV in the current time step.

Step 2 determines if the projectile has crossed the plane that contains a given UAV surface. This step is illustrated in Figure 6 below.


STEP 2
CURRENT PROJECTILE POSITION
p2
p3
INFINITE PLANE
p1
p4
PROJECTILE IS TESTED TO SEE IF IT CROSSED A PLANE USING NORMAL VECTOR DERIVED FROM THREE POINTS THAT LIE IN THE PLANE
PREVIOUS PROJECTILE POSITION

$$\text{Plane Crossed} = \left\{ \begin{array}{l} \text{True, } \operatorname{sign}(\vec{n} \bullet \vec{v}_{prev}) \neq \operatorname{sign}(\vec{n} \bullet \vec{v}_{curr}) \\ \text{False, otherwise} \end{array} \right\}$$

where $\vec{n}$ = normal to geometric plane or $\vec{v}_{14} \times \vec{v}_{12}$

$\vec{v}_{prev}$ = vector from p1 to previous position

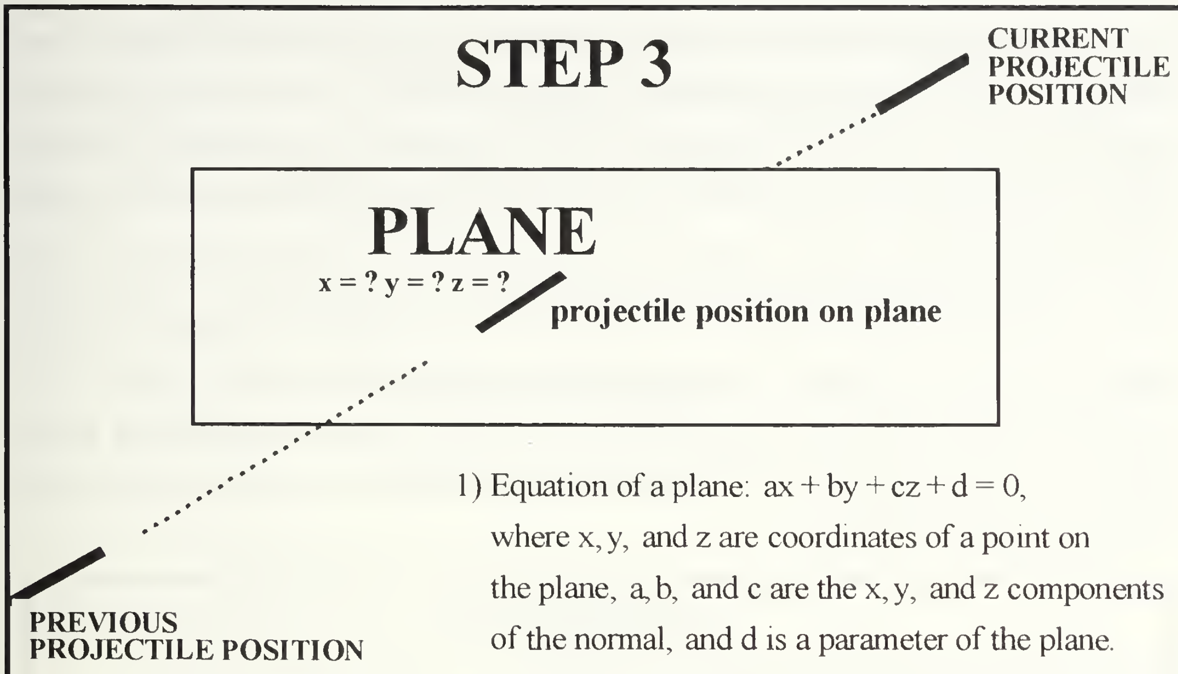
$\vec{v}_{curr}$ = vector from p1 to current position

$\vec{v}_{14}$ = vector from p1 to p4

$\vec{v}_{12}$ = vector from p1 to p2

Figure 6: Step 2 of Hit Algorithm

Step 2 determines if the projectile's previous position and current position lie on different sides of the plane which contains the UAV surface. Points p1, p2, p3 and p4 are arbitrary points in the plane containing the UAV surface. The normal vector for the plane is determined by computing the cross product of two vectors in the plane. The dot product of the normal of the plane and the vector formed by the previous projectile position and point p1 is compared with the dot product of the normal of the plane and the vector formed by the current projectile position and the point p1. If the comparison reveals that these two values have the same sign, then the projectile did not cross the plane.


STEP 3
CURRENT PROJECTILE POSITION
PLANE
x = ? y = ? z = ?
projectile position on plane
PREVIOUS PROJECTILE POSITION

1) Equation of a plane: $ax + by + cz + d = 0$,
where x, y, and z are coordinates of a point on the plane, a, b, and c are the x, y, and z components of the normal, and d is a parameter of the plane.

- $\vec{n}$ known from previous step $\Rightarrow$ values for a, b, and c.
- A point p1 is on the plane
$\Rightarrow$ $d = ax + by + cz$ where x, y, and z come from p1.

2) Parametric equation for projectile path:

$\vec{b} = \vec{b}_p + k(\vec{b}_c - \vec{b}_p)$ where $0 \leq k \leq 1$

where $\vec{b}_p$ = previous projectile position

$\vec{b}_c$ = current projectile position

Equation 2 is substituted into equation 1 to solve for the k where the bullet crosses the plane. This value for k is used in equation 1 to find the point where the bullet crosses the plane.

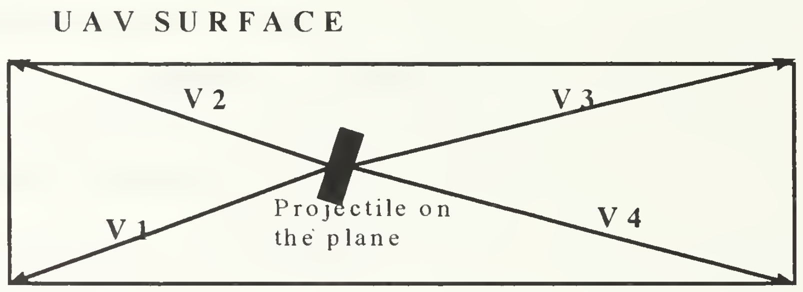
Figure 7: Step 3 of Hit Algorithm

Step 3 determines the X, Y and Z coordinate of the bullet's impact on the plane by assuming a straight line path between the previous position and the current position and using the normal computed in Step 2. This step is illustrated Figure 7 above.

Step 3 requires numerous calculations which use the vector from the previous projectile position to its current position and the normal vector to solve for the point on the plane along the path of the projectile. This result will also be used in Step 4 to determine if the point of impact on the plane lies inside the boundaries of the UAV surface being tested for penetration.

Step 4 simply computes the vectors from the impact point of the projectile on the plane to each corner point of the UAV surface. If the sum of the angles between the vectors is equal to two times pi, then the projectile must be inside the UAV surface and is classified as a hit. If the sum is not equal to two times pi, then the projectile lies on the boundary of or outside of the UAV surface and is classified as a miss. This step is illustrated in Figure 8 below.

STEP 4



If angles between V1 and V2, V2 and V3, V3 and V4, V4 and V1 sum up to 360 degrees then the projectile is inside the boundaries of the UAV surface.

Angles between vectors are computed as:

$$\vec{v}_1 \bullet \vec{v}_2 = |\vec{v}_1||\vec{v}_2| \cos\theta$$

$$\theta = a\cos\left|\frac{\vec{v}_1 \bullet \vec{v}_2}{|\vec{v}_1| \quad |\vec{v}_2|}\right|$$

Figure 8: Step 4 of Hit Algorithm

The formulas used in steps 1 to 4 are coded in Lisp in Appendix D.

G. CHARACTERISTICS OF THE PERFECT GUNNER

The modeled perfect gunner provides insight into the potential effectiveness of a projectile against a UAV. The perfect gunner can successfully engage the target with the highest probability of hit and kill, given that the object to be shot at is in range and the gunner knows its speed, altitude, and has an operational gun. The perfect gunner takes aim with all data known about the target to be engaged and is limited solely by the characteristics of the projectile, its launcher, and the algorithms used to determine the aim point to fire upon.

Due to the difficulty associated with engaging aerial targets, a high volume of fire should be established in front of the target [Ref. 20]. The perfect gunner fires five projectiles per second at an aim point in front of the UAV. This aim point is determined by two algorithms:

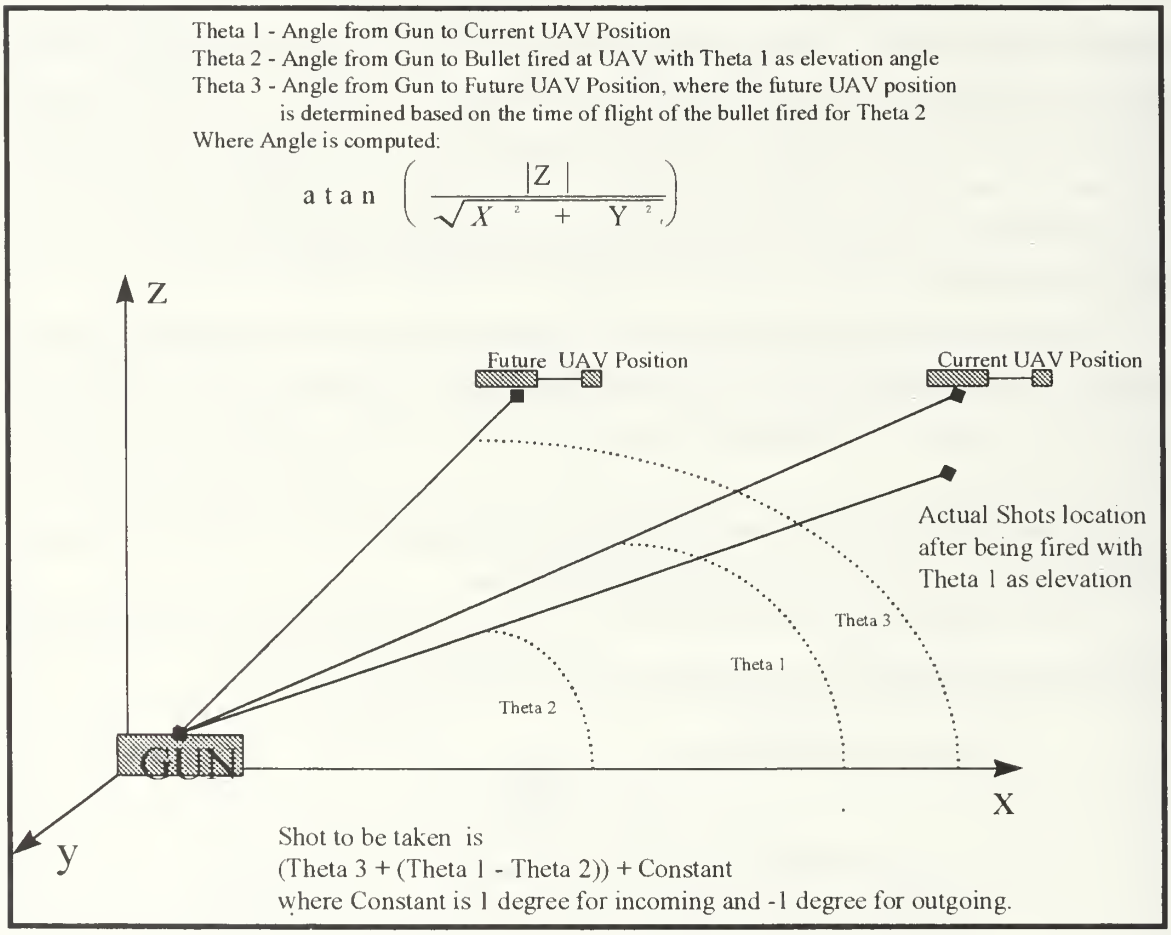
1. an "Adjust Fire Elevation" algorithm that computes the elevation (vertical angle of fire) needed to hit the UAV at its current altitude, and

2. an "Adjust Fire Azimuth" algorithm that computes the azimuth (horizontal angle of fire) needed to ensure the wall of fire is in front of the UAV.

1. Adjust Fire Elevation

a. 0 Degree Offset Adjust Fire Elevation

This algorithm has three steps and the code is found in Appendix E. Step 1 aims the gun at the current UAV position and computes the angle (Theta 1) as if the projectile travels on a straight path. Step 2 computes the angle (Theta 2) from the gun to the actual projectile location at the range to the target. This actual location is determined by firing a spotting round at the current UAV position and taking the coordinates when it passes the current UAV down range position. Step 3 computes the angle (Theta 3) to the future UAV position accounting for the time of flight of the projectile. This angle (Theta 3) is added to the difference between Theta 1 and Theta 2. The result is an elevation that takes into account the drop error of the spotting round and the time of flight. Test results indicated that this elevation angle should be further adjusted to determine the angle needed to ensure the projectile would be in front of the UAV. After preliminary testing of the algorithm, a one degree (positive for inbound and negative for outbound) super elevation constant was chosen to ensure the projectile is in front of the UAV. Figure 9 below graphically depicts Step 1 - 3. Code for this algorithm is listed in Appendix D.


Theta 1 - Angle from Gun to Current UAV Position
Theta 2 - Angle from Gun to Bullet fired at UAV with Theta 1 as elevation angle
Theta 3 - Angle from Gun to Future UAV Position, where the future UAV position is determined based on the time of flight of the bullet fired for Theta 2
Where Angle is computed:

$$\operatorname{atan}\left(\frac{|Z|}{\sqrt{X^2 + Y^2}}\right)$$

Shot to be taken is
(Theta 3 + (Theta 1 - Theta 2)) + Constant
where Constant is 1 degree for incoming and -1 degree for outgoing.

Figure 9: Adjust Fire Elevation

b. 45 Degree Offset Adjust Fire Elevation

This algorithm has 2 steps and the code is found in Appendix E. Step 1 aims the gun at the future UAV position and computes the straight line angle (Theta 3 from Figure 9 above) for this shot. Step 2 fires a series of test projectiles at increasing increments (1/2 degree per step) until a test projectile's old position and current position lie on opposite sides of the Z plane of the UAV. This elevation is then used and the gunner establishes a wall of fire on the path of the UAV.

2. Adjust Fire Azimuth

The azimuth from the gun to the UAV is computed by determining the azimuth from gun to the future UAV position. This calculation uses the future position of the UAV computed in the Adjust Fire Algorithm. In the simple case of Zero Degree offset engagements the azimuth will be either 0 or 180 degrees. Figure 10 illustrates the adjust fire azimuth algorithm.

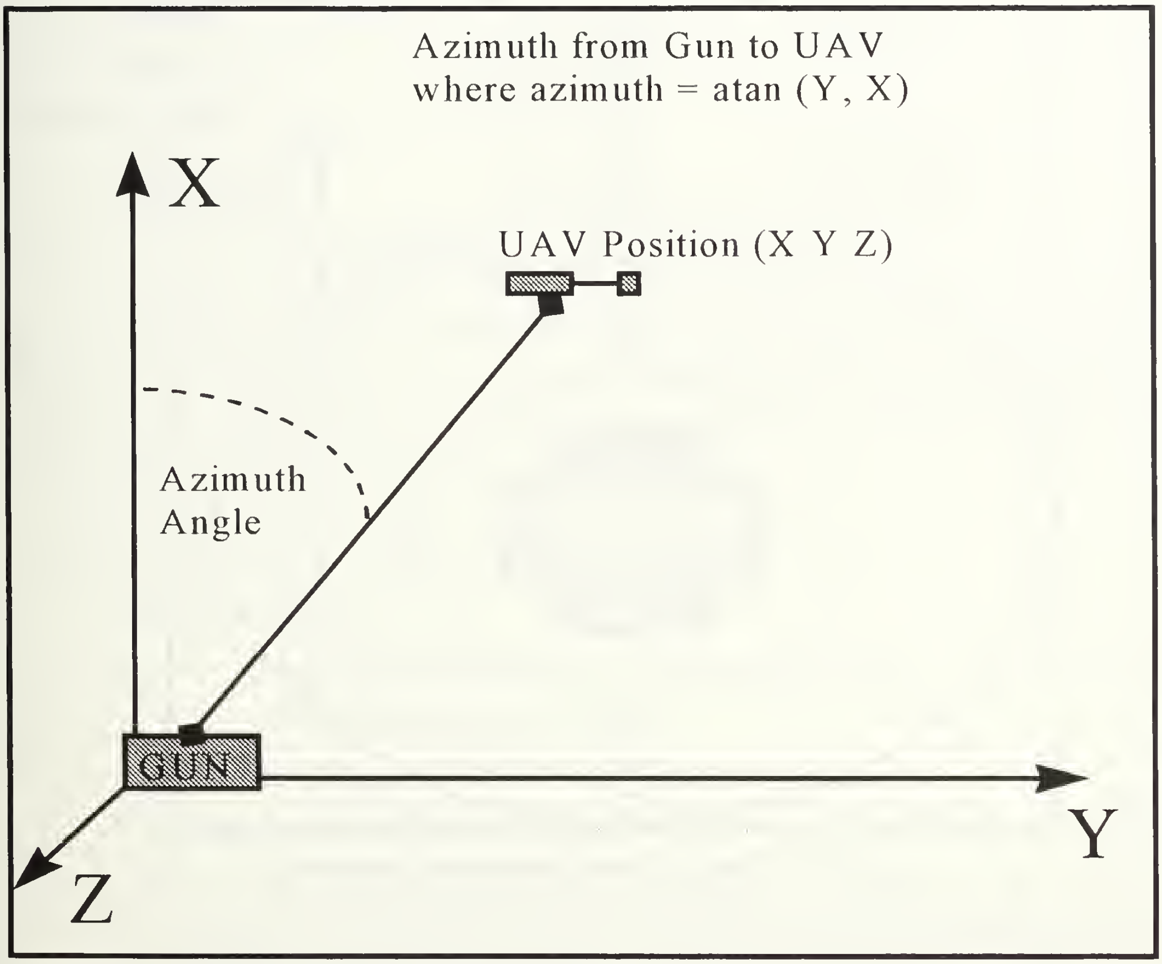


Figure 10: Adjust Fire Azimuth

H. FLIGHT PROFILES ANALYZED

Two flight profiles were considered in the tests using the perfect gun. These profiles ensure that at least one or more in-range engagements are possible. The following profiles are analyzed:

- 0 degree offset inbound, outbound UAV
- 45 degree offset crossing UAV

Figure 11 below illustrates graphically the three flight profiles at 2500 feet and 3500 feet and the engagement footprint (the area between maximum and minimum gun ranges).

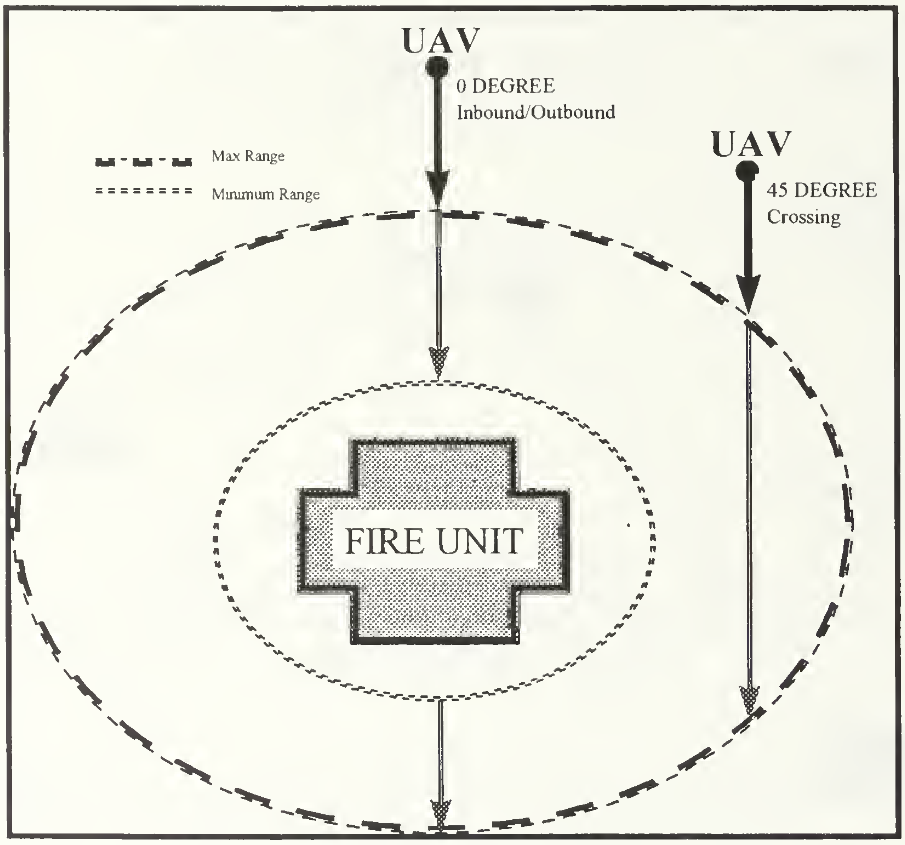


Figure 11 - Flight Profiles and Engagement Footprint

The flight profiles were run first with the perfect gunner taking shots within the appropriate minimum and maximum ranges of the gun. The results are described in what follows to allow conclusions to be drawn about the effectiveness of the modeled projectile and aiming method, and to evaluate the model in general. This is intended to provide insights into the potential effectiveness of the projectile against a UAV.

The flight profiles begin at the edge of the engagement footprint and are analyzed while inside the footprint. Table 12 below illustrates the ranges of the Gun, which has a 6100 foot effective projectile range and an elevation limit of 60 degrees for UAVs at 2500 and 3500 feet.

UAV altitude	Minimum Range of Gun	Maximum Range of Gun
0 degree inbound, outbound		
2500 feet	1443 feet	5564 feet
3500 feet	2020 feet	4996 feet
45 degree crossing		
2500 feet	1443 feet	5564 feet
3500 feet	2020 feet	4996 feet

Table 12: Ranges of Gun for flight profiles

The flight profiles are analyzed until the UAV has flown from maximum range to minimum range for incoming UAVs and from minimum range to maximum range for outgoing UAVs. Crossing Shots are evaluated from acquisition at maximum range to loss of acquisition at maximum range (from left to right).

I. LIMITATIONS AND ASSUMPTIONS

- PROJECTILES:
  - are only effected by gravity and head on drag forces;
  - are ineffective beyond 6100 feet;
  - fly a straight line between 1/10th second updates;
  - do not detonate upon impact of a UAV surface because of the soft nature of the UAVs surfaces;
  - have a random muzzle velocity of 3608 +/- 16.4 feet per second;
  - have an angular dispersion of .05006219 degrees.

- UAV:
  - flies at a constant altitude (2500 or 3500 feet) and

    - flies at a constant speed (65 knots).

- GUNNER:
    - detects all UAVs in the engagement footprint,
    - can accurately determine the speed and altitude of the UAV,
    - is oriented on the target line prior to the UAV crossing the engagement footprint,
    - is only posed with one threat UAV at a time,
    - is oriented on the target line for 0 degree incoming/outgoing shots and oriented near (within 223 feet) the target line for 45 degree offset simulations,
    - has unlimited ammunition.

J. SUMMARY

This chapter presents a complete simulation model with a scenario similar to that of a BSFV in the Air Defense role, a UAV with characteristics similar to that of a Pioneer class UAV, and a projectile and launcher modeled after the BSFV turret and the M792 25mm Heat round. Algorithms developed to determine the penetration of any UAV surface by the projectile and to aim in a manner similar to that of doctrinal US Army firing techniques allow for testing of the Modeled Launcher and Projectile with aiming capabilities against the simulated UAV using the profiles described earlier. Chapter V reviews each profile, displays the results, and offers an analysis of each run.

V. RESULTS

A. 0 DEGREE OFFSET INBOUND/OUTBOUND

1. Introduction

The following tables show the results obtained using the simulation described in Chapter IV of a BSFV against the UAV flying the 0 degree offset inbound/outbound flight profile. The tables show the acquired range where the perfect gunner establishes his aim and commences firing, and the cease fire range where the perfect gunner stops engaging due to the UAV passing beyond the wall of fire. At each acquired range it is assumed that the target has no damage. The simulation results shown included randomly generated angular and velocity dispersions, so repeat runs will give different results. However, these results are typical of results obtained from other runs.

Acquired Range	Cease Fire Range	Shots Fired	Body Hit	Wing Hit	Tail Hit	[illegible] Hit	[illegible]	Total Hits
1443	1577	16	2	0	2	0	0	4
1577	1719	12	2	0	0	0	0	2
1719	1869	13	2	0	0	0	0	2
1869	2019	13	2	0	0	1	0	3
2019	2181	14	2	0	0	0	0	2
2181	2349	15	2	1	0	0	0	3
2349	2528	16	2	0	0	0	0	2
2528	2720	17	0	1	0	0	0	1
2720	2924	18	0	0	0	0	0	0
2924	3146	19	2	0	0	0	0	0
3146	3380	20	2	1	0	0	0	3
3380	3632	22	4	0	0	0	0	4
3632	3908	24	2	0	0	0	0	2
3908	4196	25	0	1	0	0	0	1
4196	4508	27	2	0	0	0	0	2
4508	4838	28	0	0	0	0	0	0
4838	5192	30	0	1	0	2	0	3
5192	Max	33	0	2	0	0	0	2

Table 13: 2500 Foot Altitude, 0 Degree Offset Outbound Simulation

Engaged Range	Cease Fire Range	Shots Fired	Dome Hit	Wing Hit	Tail Hit	Fuselage Hit	Fin Hit	Total Hits
5564	4867	63	0	0	0	1	0	0
4867	4310	47	0	0	0	0	0	0
4310	3860	38	2	0	0	0	0	2
3860	3482	32	4	0	1	0	0	5
3482	3158	28	0	1	1	0	0	2
3158	2876	24	2	0	0	0	0	2
2876	2624	21	2	0	1	0	0	3
2624	2396	19	0	1	0	0	0	1
2396	2192	17	2	0	0	0	0	2
2192	2000	16	2	0	0	0	0	2
2000	1820	15	2	0	0	0	0	2
1820	1652	14	2	0	1	1	0	4
1652	1496	13	2	1	1	0	0	4
1496	Min	3	1	0	0	0	0	1

Table 14: 2500 Foot Altitude, 0 Degree Offset Inbound Simulation

Engaged Range	Cease Fire Range	Shots Fired	Dome Hit	Wing Hit	Tail Hit	Fuselage Hit	Fin Hit	Total Hits
4996	4491	47	2	0	0	0	0	2
4491	4053	37	2	0	0	0	0	2
4053	3664	33	0	0	0	0	0	0
3664	3328	28	0	1	1	0	0	2
3328	3022	26	4	0	1	0	0	5
3022	2740	24	2	0	0	0	0	2
2740	2482	22	2	1	0	0	0	3
2482	2248	20	2	0	0	0	0	2
2248	2020	19	2	1	0	0	0	3
2020	Min	0						

Table 15: 3500 Foot Altitude, 0 Degree Offset Inbound Simulation

Acquired Range	Cease Fire Range	Shots Fired	Body Hit	Wing Hit	Tail Hit	Extension Hit	Fin Hit	Total Hits
2020	2200	20	2	0	0	0	0	2
2200	2386	16	0	0	1	0	0	1
2386	2776	17	0	0	1	0	0	1
2776	2979	17	2	0	0	0	0	2
2979	3195	18	0	0	0	0	0	0
3195	3429	20	2	0	0	0	0	2
3429	3675	21	4	0	1	0	0	5
3675	3933	22	2	0	0	0	0	2
3933	4191	23	0	0	0	0	0	0
4191	4467	23	0	1	0	0	0	1
4467	4761	25	0	0	1	0	0	1
4761	Max	22	0	0	0	0	0	0

Table 16: 3500 Foot Altitude, 0 Degree Offset Outbound Simulation

2. Analysis

The tables clearly show that the simulated gun and projectile are capable of penetrating the simulated UAVs surfaces in most engagement sequences (only 9 of 54 engagement sequences resulted in 0 hits). Clearly the abundance of body hits are a result of firing on the target line and the relatively large number of body surfaces (top body, bottom body, left body, right body, nose and rear). Similarly, the lack of any success against the UAVs fins are a result of the fin not being exposed on an inbound/outbound simulation run. From this, it can be concluded that the rate of fire and aiming method coupled with the characteristics of the projectile are indeed adequate for engaging the UAV inside the range of the projectile.

B. 45 DEGREE OFFSET CROSSING

1. Introduction

The following tables show the results obtained using the simulation described in Chapter IV of the BSFV against the UAV flying the 45 degree offset crossing flight profile. The tables show the acquired range where the perfect gunner establishes his aim and commences firing, and the cease fire range where the perfect gunner stops engaging due to the UAV passing beyond the wall of fire. At each acquired range it is again assumed that the target has no damage. As in the 0 degree offset cases, these simulation results also included randomly generated angular and velocity dispersions, so repeat runs will give different, although similar, results.

Acquired Range	Cease Fire Range	Offset Angle at Acquisition	Shots Fired	Body Hit	Wing Hit	Tail Hit	Explosion Hit	Fin Hit	Total Hits
4330	4244	45	18	0	0	1	0	0	1
4244	4167	47	18	0	0	0	1	0	1
4167	4092	49	13	0	1	0	0	0	1
4092	4022	52	13	0	0	0	0	0	0
4022	3955	54	13	0	0	0	1	0	1
3955	3895	56	12	2	0	0	0	0	2
3895	3839	59	12	2	0	0	0	1	3
3839	3790	61	12	0	0	0	1	1	2
3790	3742	64	12	0	0	1	1	0	2
3742	3701	67	11	2	0	1	0	0	3
3701	3664	69	11	0	1	0	0	0	1
3664	3632	72	11	2	0	0	0	0	2
3632	3605	75	11	0	0	1	0	0	1
3605	3582	77	11	2	0	0	0	0	2
3582	3564	79	11	0	0	0	0	1	1
3564	3550	83	11	0	0	0	0	0	0
3550	3540	86	11	2	0	0	0	0	2
3540	3536	89	11	0	0	0	0	1	1
3536	3536	92	12	0	0	0	1	0	1
3536	3540	95	12	2	0	0	0	0	2
3540	3549	98	12	0	0	0	1	0	1
3549	3563	101	11	0	1	0	0	0	1
3563	3582	104	12	0	0	0	0	0	0
3582	3606	106	12	2	0	0	0	0	2
3606	3634	109	12	2	1	0	0	0	3
3634	3667	112	12	0	0	1	0	0	1
3667	3704	113	12	2	0	0	0	0	2
3704	3748	116	12	2	0	0	0	0	2
3748	3796	119	12	2	0	0	0	0	2
3796	3846	121	12	0	0	0	0	0	0
3846	3902	123	12	0	1	1	0	0	2
3902	3962	126	12	2	0	1	0	1	4
3962	4029	128	13	2	0	0	1	0	3
4029	4100	130	12	2	0	0	0	0	0
4100	4178	132	13	0	0	0	0	1	1
4178	4260	133	13	0	0	0	0	1	1
4260	4349	135	14	0	0	0	0	1	1

Table 17: 2500 Foot Altitude, 45 Degree Offset Crossing Simulation

[illegible]	[illegible]	[illegible]	Shots Fired	[illegible]	[illegible] Hit	[illegible]	[illegible] Hit	[illegible]	[illegible]
4730	4662	45	17	0	1	0	0	0	1
4662	4600	47	12	0	0	0	1	0	1
4600	4543	50	12	0	0	1	0	0	1
4543	4489	53	12	0	1	0	0	0	1
4489	4441	55	11	0	0	0	0	0	0
4441	4397	57	11	2	0	0	0	0	2
4397	4356	60	11	2	0	0	0	0	2
4356	4318	63	11	2	0	0	0	0	2
4318	4285	65	11	2	0	0	0	0	2
4285	4257	68	11	0	1	0	1	0	2
4257	4232	71	11	2	1	0	0	0	3
4232	4210	74	11	0	0	0	0	0	0
4210	4192	77	11	0	1	0	0	0	1
4192	4169	80	11	0	1	0	0	0	1
4169	4163	83	11	0	0	0	0	1	1
4163	4160	86	11	0	1	0	0	0	1
4160	4162	90	11	0	0	1	0	0	1
4162	4167	93	11	2	0	0	0	0	2
4167	4176	96	11	0	0	0	0	0	0
4176	4189	99	11	2	0	0	0	0	2
4189	4206	103	11	2	1	0	0	0	3
4206	4226	105	11	2	0	0	1	0	3
4226	4250	108	11	2	0	1	0	0	3
4250	4279	111	11	2	1	0	0	0	3
4279	4312	114	11	0	0	0	0	1	1
4312	4349	117	11	2	0	0	0	0	2
4349	4389	120	11	2	0	1	0	0	3
4389	4435	122	12	2	0	1	1	0	4
4435	4484	125	12	0	0	0	0	0	0
4484	4358	127	12	0	0	0	1	1	2
4358	4597	129	13	0	0	1	0	0	1
4597	4660	131	13	0	0	0	0	0	0
4660	4727	135	13	0	1	0	0	0	1

Table 18: 3500 Foot Altitude, 45 Degree Offset Crossing Simulation

2. Analysis

The tables clearly show that the simulated gun and projectile are capable of penetrating the simulated UAVs surfaces in most engagement sequences (only 10 of 70 engagement sequences resulted in 0 hits). Clearly the success of engaging the UAV in all parts is due to the fact that as the UAV crosses, its wings, extensions, body and tail each are exposed to the wall of fire. Once again, the simulation establishes the capability of the projectile to successfully penetrate the UAV given the aiming method and rate of fire utilized.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The goal of this thesis was to develop a method to determine the ability of the Bradley Stinger Fighting Vehicle (BSFV) 25mm Cannon to counter the emerging threat of Unmanned Aerial Vehicles (UAVs). The modeled objects all inherit unclassified characteristics of the BSFV, one of its primary ammunition rounds, and a representative UAV. Although many assumptions were made to simplify the model, simulation runs clearly demonstrate that the rate of fire utilized in the simulation along with the aiming system of placing a wall of fire in front of the target allow the projectile to penetrate the surface of the UAV in 125 out of 154 independent runs of inbound, outbound and crossing patterns. From these tests, it would seem that the modeled BSFV, given assumptions made about the gunner's aiming method and acquisition capability, is in fact capable of successfully engaging the modeled Pioneer class UAV. This conclusion, however, is only relative to this model and may not be substantiated in actual physical experimentation. Clearly, the most questionable assumption in the simulation is that the gunner can acquire the UAV and orient the gun on the target line at maximum range. One critical, and perhaps most unrealistic, implication made by this assumption is the ability of the gunner to aim the system accurately in front of the UAVs flight path. However, in the author's opinion, it is not unrealistic to assume that today's research and development efforts may in fact allow that capability to become a reality in the near future through the development of appropriate training aids to the gunner coupled with advances in aiming and detection systems.

B. RECOMMENDATIONS FOR FUTURE RESEARCH

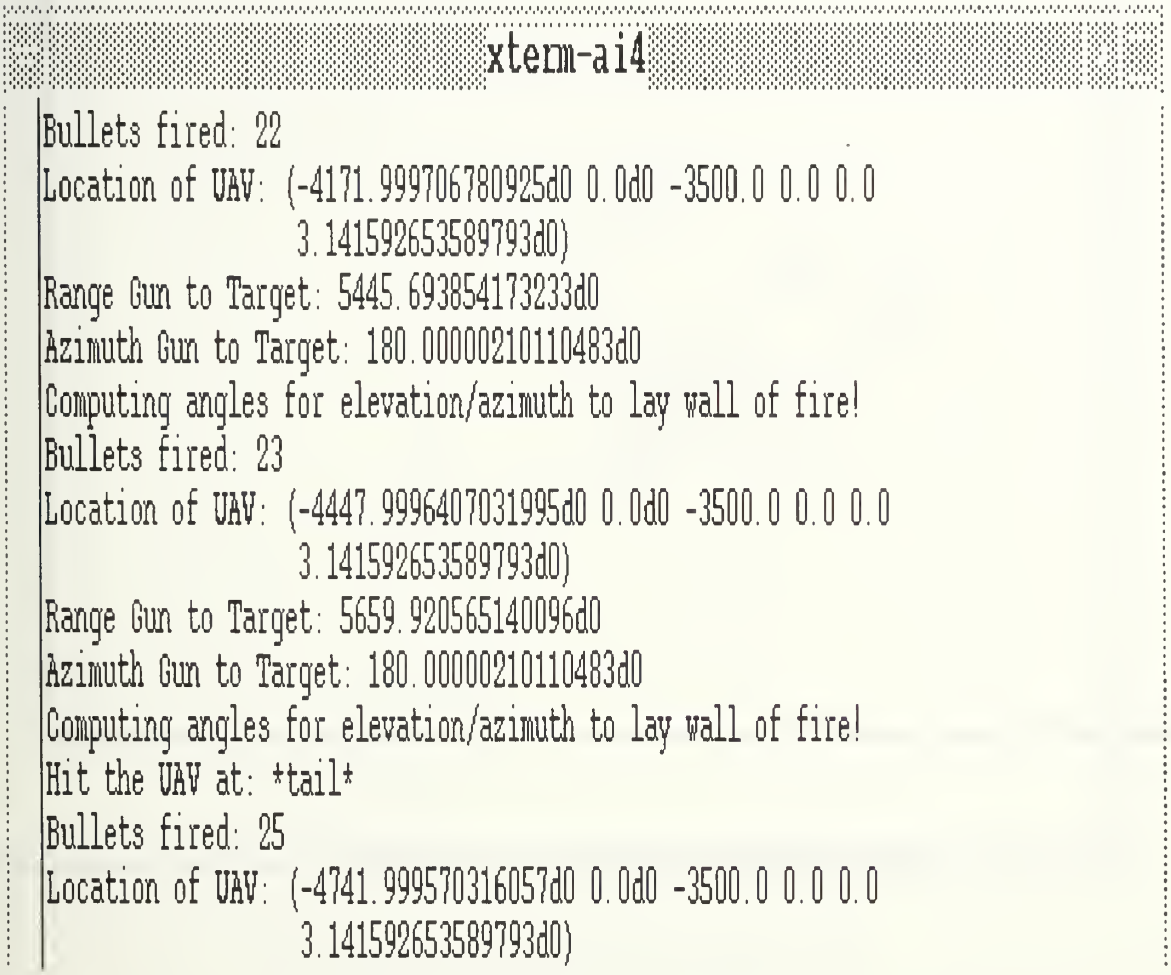
There are many ways to improve upon this methodology or to build upon it in its current form. Much work remains to be accomplished regarding the number of engagements a system can expect to get with the current human acquisition capability and sensor acquisition. Additionally, the effects of nature on all facets of the model could be added. To do this it would be necessary to bring into bear the effects of weather, terrain, vision and human ability into the model. In its current state, the model can provide some insight into the probability of hit and kill based on some analysis of the surface area vulnerabilities to the projectile in use and the results obtained in Chapter V. The source code and algorithms developed in this research could be improved upon to work in a more efficient and possibly more realistic manner, especially if used with classified data regarding the gun and target.

Several simplifying assumptions were made during the design and implementation of the simulation in order to allow the author to complete and test a fully functional methodology. Each assumption should be examined and relaxed to better reflect a true representation of the modeled objects.

Lastly, it would be useful for this model to be ported to a desktop Personal Computer (PC) which would involve rewriting Appendix B (Camera Code) to conform to a PC platform.

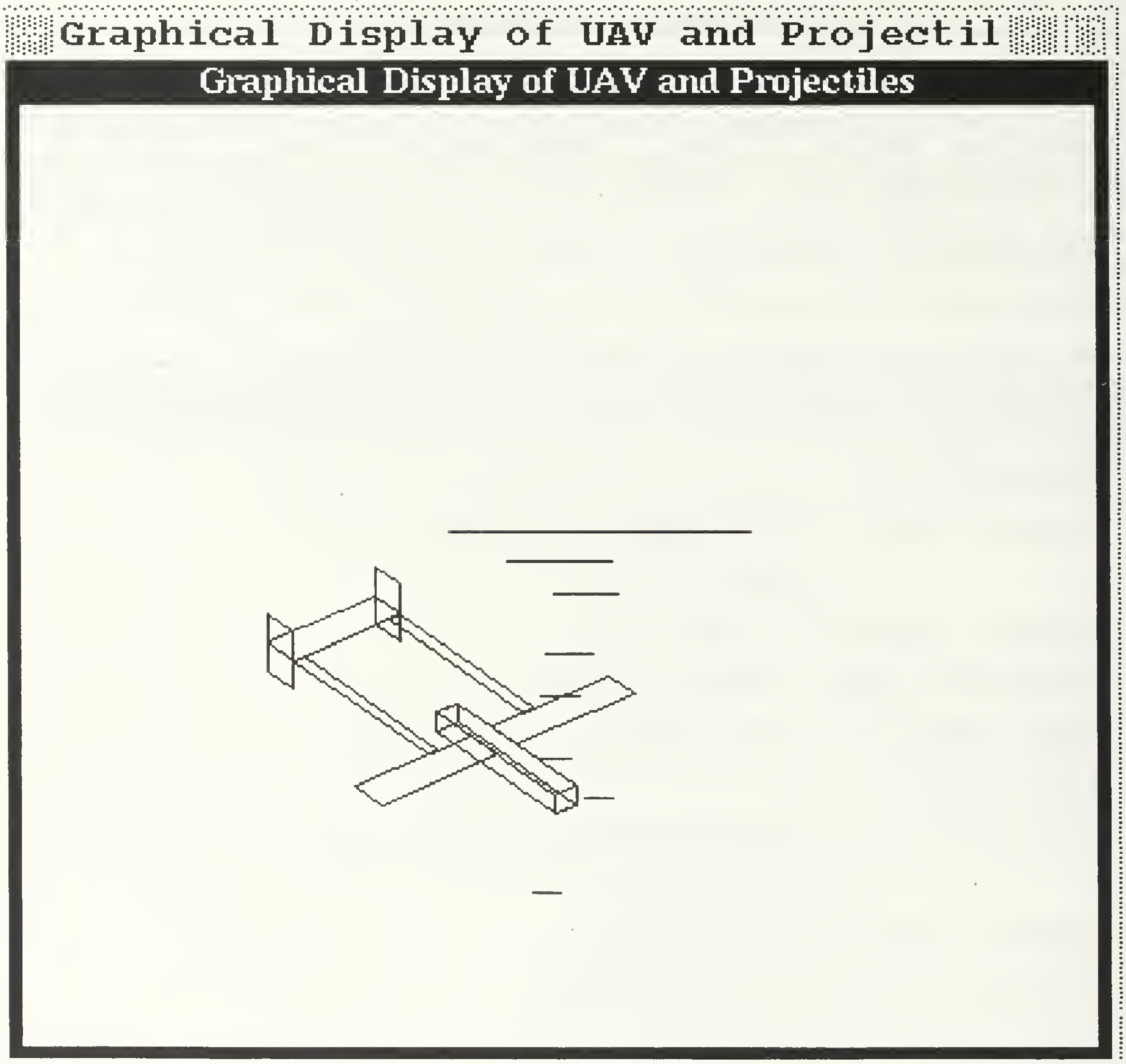
APPENDIX A: SCREEN IMAGES

The following screen images provide an example of the output generated during a simulation run with the UAV flying a crossing, inbound and outbound profile. The projectiles are represented by a horizontal line which allows for depth perception; the wider the line the closer the projectile is to the gun.

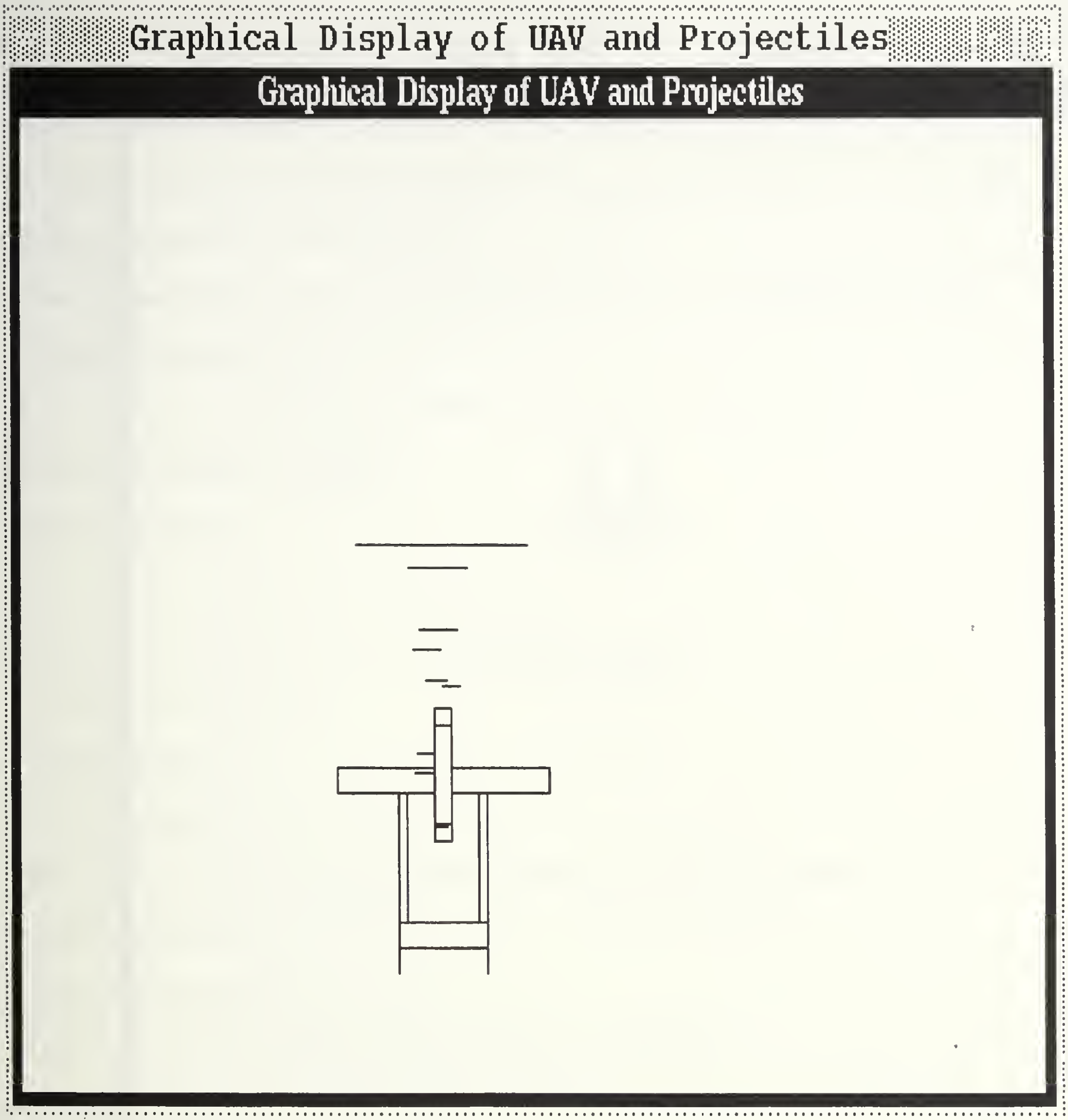


```
Bullets fired: 22
Location of UAV: (-4171.999706780925d0 0.0d0 -3500.0 0.0 0.0
                  3.141592653589793d0)
Range Gun to Target: 5445.693854173233d0
Azimuth Gun to Target: 180.00000210110483d0
Computing angles for elevation/azimuth to lay wall of fire!
Bullets fired: 23
Location of UAV: (-4447.9996407031995d0 0.0d0 -3500.0 0.0 0.0
                  3.141592653589793d0)
Range Gun to Target: 5659.920565140096d0
Azimuth Gun to Target: 180.00000210110483d0
Computing angles for elevation/azimuth to lay wall of fire!
Hit the UAV at: *tail*
Bullets fired: 25
Location of UAV: (-4741.999570316057d0 0.0d0 -3500.0 0.0 0.0
                  3.141592653589793d0)
```

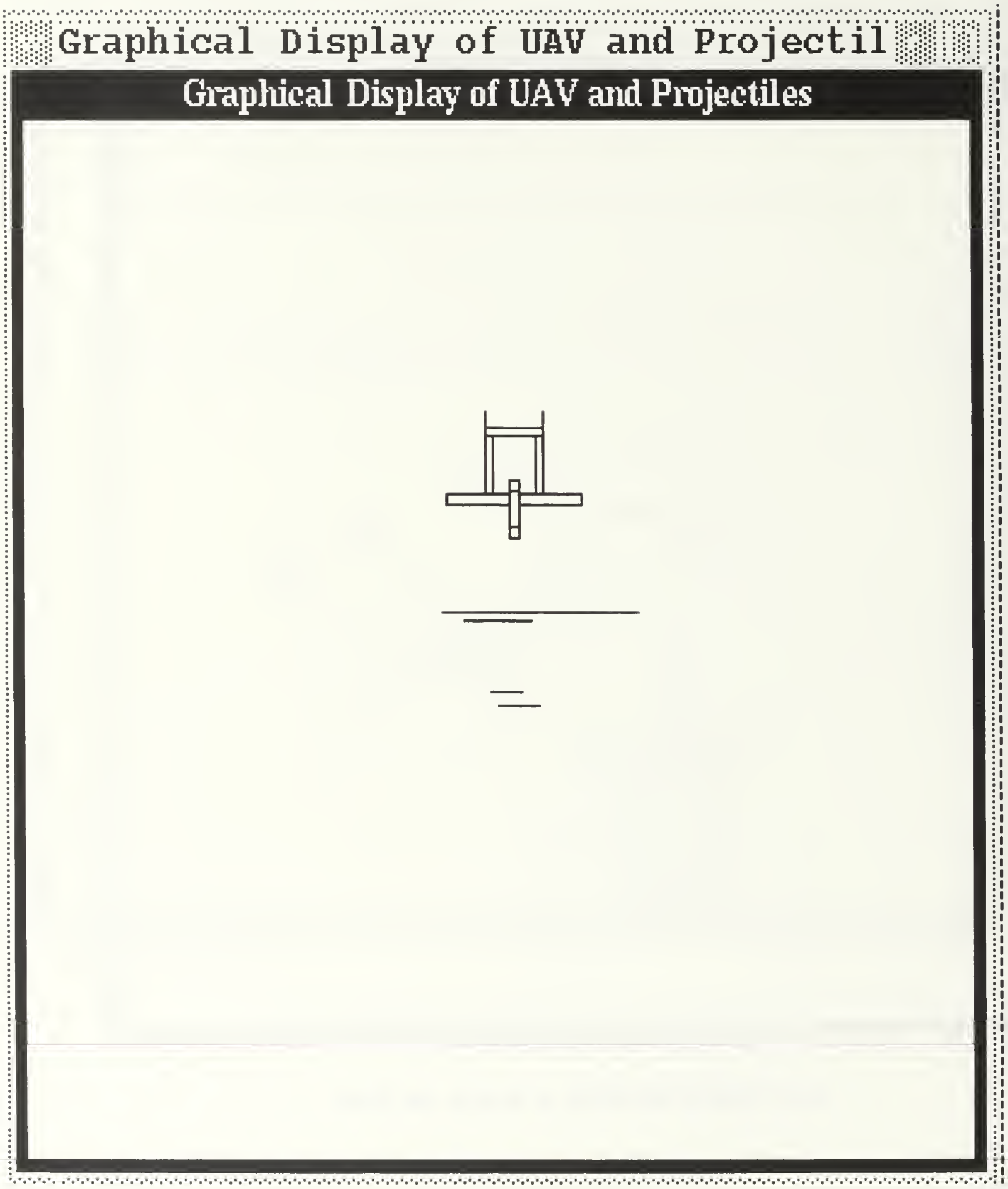
SAMPLE OUTPUT DURING A SIMULATION RUN


Graphical Display of UAV and Projectil
Graphical Display of UAV and Projectiles

UAV CROSSING INTO WALL OF FIRE


Graphical Display of UAV and Projectiles
Graphical Display of UAV and Projectiles

UAV INBOUND INTO A WALL OF FIRE


Graphical Display of UAV and Projectil
Graphical Display of UAV and Projectiles

UAV OUTBOUND INTO A WALL OF FIRE

APPENDIX B: SOURCE CODE (UAV AND PROJECTILE FILES)

```
; File: pioneer-euler-angle-rigid-body.cl      Franz Common LISP

; ** RIGID BODY CLASS DEFINITION **

; Defines a Pioneer UAV like rigid body

; Code written by R.B. McGhee, Naval Postgraduate School, & modified by D. Wiley

; mcghee@cs.nps.navy.mil

;*****************************************************************

(defconstant *gravity* 32.2185)

(defclass rigid-body

   ()

  ((posture     ;The vector (xe ye ze phi theta psi).

    :initform '(-200 0 0 0 0 0)

    :initarg :posture

    :accessor posture)

   (posture-rate ;The vector (xe-dot ye-dot ze-dot phi-dot theta-dot psi-dot).

    :initarg :posture-rate

    :accessor posture-rate)

   (velocity    ;The six-vector (u v w p q r) in body coordinates.

    :initform '(0 0 0 0 0 0)

    :initarg :velocity

    :accessor velocity)

   (velocity-growth-rate  ;The vector (u-dot v-dot w-dot p-dot q-dot r-dot).

    :accessor velocity-growth-rate)

   (forces-and-torques    ;The vector (Fx Fy Fz L M N) in body coordinates.
 :initform (list 0 0 (- *gravity*) 0 0 0)
 :accessor forces-and-torques)
(moments-of-inertia    ;The vector (Ix Iy Iz) in principal axis coordinates.
 :initform '(1 1 1)
 :initarg :moments-of-inertia
 :accessor moments-of-inertia)
(mass
 :initform 1
 :initarg :mass
 :accessor mass)
(node-list    ;(x y z 1) in body coord for each node. Starts with (0 0 0 1).
 :initform '((0 0 0 1)(9 -1 0 1)(9 1 0 1)(-5 -1 0 1)(-5 1 0 1)     ;0-4
    (9 -1 2 1)(9 1 2 1)(-5 -1 2 1)(-5 1 2 1)(2 -1 0 1)          ;5-9
    (2 1 0 1)(-1 -1 0 1)(-1 1 0 1)(2 -12 0 1)(2 12 0 1)         ;10-14
    (-1 -12 0 1)(-1 12 0 1)(-17 -5 -3 1)(-17 -5 3 1)(-20 -5 -3 1);15-19
    (-20 -5 3 1)(-1 -5 0 1)(-1 -4 0 1)(-17 -5 0 1)(-17 -4 0 1)   ;20-24
    (-17 5 -3 1)(-17 5 3  1)(-20 5 -3 1)(-20 5 3 1)(-1 4 0 1)    ;25-29
    (-1 5 0 1)(-17 4 0 1)(-17 5 0 1)(-20 -5 0 1)(-20 5 0 1))     ;30-34
 ;Defines a "pioneer UAV" as default rigid body.
 :initarg :node-list
 :accessor node-list)
(polygon-list
 :initform '(
         ; comments read as looking from the "top" at the "body"
         (1 2 4 3) ;top body
```

```
          (1 2 6 5) ;front nose
          (1 3 7 5) ;side left_body
          (2 4 8 6) ;side right_ body
          (5 6 8 7) ;bottom body
          (9 13 15 11) ;top left wing
          (10 14 16 12);top right wing
          (3 4 8 7)  ;rear body
          (17 18 20 19) ; left tail fin
          (25 26 28 27) ;right tail fin
          (21 22 24 23) ; bottom left extender
          (29 30 32 31) ;bottom right extender
          (23 32 34 33) ;bottom tail
          )
   :initarg :polygon-list
   :accessor polygon-list)
  (transformed-node-list ;(x y z 1) in earth coord for each node in node-list.
   :accessor transformed-node-list)
  (H-matrix
   :initform (unit-matrix 4)
   :accessor H-matrix)
  (time-stamp
   :accessor time-stamp)))
(defmethod initialize ((body rigid-body))
 (setf (transformed-node-list body) (node-list body))
 (setf (velocity-growth-rate body) (update-velocity-growth-rate body))
 (setf (posture-rate body) (earth-velocity body))
 (setf (time-stamp body) (get-internal-real-time)))
(defmethod update-posture ((body rigid-body) delta-t) ;Euler integration.
  (setf (posture-rate body) (earth-velocity body))
  (setf (posture body)
  (vector-add (posture body) (scalar-multiply delta-t (posture-rate body)))))
(defmethod move-body ((body rigid-body) azimuth elevation roll x y z)
  (setf (posture body) (list x y z roll elevation azimuth))
  (setf (H-matrix body)
  (homogeneous-transform azimuth elevation roll x y z))
  (transform-node-list body))
(defmethod update-rigid-body ((body rigid-body))     ;Euler integration.
  (let* ((delta-t (get-delta-t body)))
        (update-posture body delta-t)
        (setf (H-matrix body) (homogeneous-transform (sixth (posture body))
         (fifth (posture body)) (fourth (posture body)) (first (posture body))
         (second (posture body)) (third (posture body))))
        (transform-node-list body)
        (update-velocity body delta-t)
        (update-velocity-growth-rate body)))
 (defmethod get-delta-t ((body rigid-body)) .1)
 (defmethod update-velocity ((body rigid-body) delta-t) ;Euler integration.
   (setf (velocity body)
   (vector-add (velocity body)
             (scalar-multiply delta-t (velocity-growth-rate body)))))
```

```
(defmethod earth-velocity ((body rigid-body))
 (let* ((linear-velocity (firstn 3 (velocity body)))
        (rotational-velocity (cdddr (velocity body)))
        (posture (posture body))
        (R-matrix (rotation-matrix (sixth posture) (fifth posture)
                                   (fourth posture)))
        (linear-earth-velocity (post-multiply R-matrix linear-velocity))
        (T-matrix (body-rate-to-euler-rate-matrix (sixth posture)
                   (fifth posture) (fourth posture)))
        (rotational-earth-velocity (post-multiply T-matrix
                                                   rotational-velocity)))
      (append linear-earth-velocity rotational-earth-velocity)))
(defmethod transform-node-list ((body rigid-body))
 (setf (transformed-node-list body)
   (mapcar #'(lambda (node-location)
              (post-multiply (H-matrix body) node-location))
          (node-list body))))
```

```
; File: bullet.cl          Franz Common LISP
;
; ** RIGID BODY CLASS DEFINITION **
; Defines a bullet like rigid body
; Code written by D. A. Wiley, Naval Postgraduate School,
; dawiley@cs.nps.navy.mil
; ***************************************************************


(defclass bullet (rigid-body)
   ((forces-and-torques
     :initform '(0 0 0 0 0 0))
    (old-posture   ;The vector (xe ye ze phi theta psi)
     :initform '(0 0 0 0 0 0)
     :initarg :old-posture
     :accessor old-posture)
    (time-of-flight
     :initform 0
     :accessor time-of-flight)
   (polygon-list
     :initform '((1 2)))))
(setf k1 0.0001)
(defmethod update-velocity-growth-rate ((body bullet))
  (let ((u-dot (- (* (- k1) (first (velocity body)) (abs (first (velocity body))))
               (* (sin (fifth (posture body))) *gravity*)))
       (w-dot (+ (* (- k1) (third (velocity body)) (abs (third (velocity body))))
                (* (cos (fifth (posture body))) *gravity*))))
  (setf (velocity-growth-rate body) (list u-dot 0 w-dot 0 0 0))))
(defun distance-bullet-plane (xp xb yp yb zp zb)
  (sqrt (+(square(- xp xb))(square(- yp yb))(square(- zp zb)))))
(defun update_bullets (bullet-list)
  (cond ((null bullet-list) nil)
      (t (update-rigid-body (first bullet-list))
        (if (relevent-bullet airplane-1 (first bullet-list))
          (setf bullet-1 (cons (first bullet-list)
                      (update_bullets (rest bullet-list))))))))
(defun relevent-bullet (plane bullet)
  (if (and (above-ground-bullet (third (posture bullet)))
        (< (distance-gun-object bullet)
          (distance-gun-object airplane-1)))t))
(defun above-ground-bullet (projectile)
  (if (< projectile 1.0) t))
(defun distance-gun-object (object)
  (let ((x (first (posture object)))(y (second (posture object)))
      (z (third (posture object))))
      (sqrt (+ (square x)(square y)(square z)))))
(defmethod update-posture ((bullet bullet) delta-t)
  (setf (posture-rate bullet) (earth-velocity bullet))
  (setf (old-posture bullet) (posture bullet))
  (setf (posture bullet)
    (vector-add (posture bullet) (scalar-multiply delta-t (posture-rate bullet)))))
```

```
; File: uav-components.cl     Franz Common LISP
;
; ** The makeup of a UAV rigid body and functions
;    for screen output, this file is used in conjuntion
;    with the determine-hit.cl for testing independently
;    the surface of a UAV **
;
; Code written by D. A. Wiley, Naval Postgraduate School,
; dawiley@cs.nps.navy.mil
; *****************************************************************


(defconstant *top-body* '((-5 -1 0)(9 -1 0)(9 1 0)(-5 1 0)))
(defconstant *nose* '((9 -1 0)(9 1 0)(9 1 2)(9 -1 2)))
(defconstant *left-body* '((9 -1 2)(9 -1 0)(-5 -1 0)(-5 -1 2)))
(defconstant *right-body* '((-5 1 2)(-5 1 0)(9 1 0)(9 1 2)))
(defconstant *bottom-body* '((9 -1 2)(9 1 2)(-5 1 2)(-5 -1 2)))
(defconstant *left-wing* '((-1 -12 0)(2 -12 0)(2 -1 0)(-1 -1 0)))
(defconstant *right-wing* '((-1 1 0)(2 1 0)(2 12 0)(-1 12 0)))
(defconstant *left-tail-extender* '((-17 -5 0)(-1 -5 0)(-1 -4 0)(-17 -4 0)))
(defconstant *right-tail-extender* '((-17 4 0)(-1 4 0)(-1 5 0)(-17 5 0)))
(defconstant *left-tail-fin* '((-20 -5 3)(-17 -5 3)(-17 -5 -3)(-20 -5 -3)))
(defconstant *right-tail-fin* '((-20 5 3)(-17 5 3)(-17 5 -3)(-20 5 -3)))
(defconstant *tail* '((-20 -5 0)(-17 -5 0)(-17 5 0)(-20 5 0)))
(defconstant *rear* '((-5 -1 2)(-5 -1 0)(-5 1 0)(-5 1 2)))
```

```
(defconstant *uav-components* (list  *top-body* *nose* *left-body*
                              *right-body* *bottom-body* *left-wing*
                              *right-wing* *left-tail-extender* *right-tail-extender*
                              *left-tail-fin* *right-tail-fin*
                              *tail* *rear*))
(defconstant *uav-components-strings* '("*top-body*""*nose*""*left-body*""*right-body*"
       "*bottom-body*""*left-wing*""*right-wing*""*left-tail-extender*"
       "*right-tail-extender*""*left-tail-fin*""*right-tail-fin*""*tail*""*rear*"))
(setf *bullet-#* 0)
(defun provide-feedback (uav-part bullet-ID)
  (cond ((equal uav-part (nth 0 *uav-components*))
        (format t "~%Hit the UAV at: ~A" (nth 0 *uav-components-strings*)))
       ((equal uav-part (nth 1 *uav-components*))
        (format t "~%Hit the UAV at: ~A" (nth 1 *uav-components-strings*)))
       ((equal uav-part (nth 2 *uav-components*))
        (format t "~%Hit the UAV at: ~A" (nth 2 *uav-components-strings*)))
       ((equal uav-part (nth 3 *uav-components*))
        (format t "~%Hit the UAV at: ~A" (nth 3 *uav-components-strings*)))
       ((equal uav-part (nth 4 *uav-components*))
        (format t "~%Hit the UAV at: ~A" (nth 4 *uav-components-strings*)))
       ((equal uav-part (nth 5 *uav-components*))
        (format t "~%Hit the UAV at: ~A" (nth 5 *uav-components-strings*)))
       ((equal uav-part (nth 6 *uav-components*))
        (format t "~%Hit the UAV at:: ~A" (nth 6 *uav-components-strings*)))
((equal uav-part (nth 7 *uav-components*))
 (format t "~%Hit the UAV at: ~A" (nth 7 *uav-components-strings*)))
((equal uav-part (nth 8 *uav-components*))
 (format t "~%Hit the UAV at: ~A" (nth 8 *uav-components-strings*)))
((equal uav-part (nth 9 *uav-components*))
 (format t "~%Hit the UAV at: ~A" (nth 9 *uav-components-strings*)))
((equal uav-part (nth 10 *uav-components*))
 (format t "~%Hit the UAV at: ~A" (nth 10 *uav-components-strings*)))
((equal uav-part (nth 11 *uav-components*))
 (format t "~%Hit the UAV at: ~A" (nth 11 *uav-components-strings*)))
((equal uav-part (nth 12 *uav-components*))
 (format t "~%Hit the UAV at: ~A" (nth 12 *uav-components-strings*)))
(t (pprint "error in providefeedback function"))))
```

APPENDIX C: SOURCE CODE (STROBE CAMERA FILE)

```
; File: camera.cl                                   Franz Common LISP

; ** CAMERA CLASS DEFINITION **

; A Camera  "takes a picture" of rigid-body class objects

; and displays the image.  A sequence of images may be

; displayed by superimposing them or by first erasing the display

; window and then creating and displaying the next image.

; Requires: rigid-body.cl

; by Shirley Isakari  CS4314 Winter 1994  Final Project

; Modifications & enhancements to Prof. McGhee's Strobe-Camera CLOS code

; *****************************************************************

(require :xcw)

(use-package :cw)  ; Note that this is required for use of mouse and color.

                   ; This forced renaming of some original functions, i.e.

                   ; move and translate. Causes some problem when compiling.

(cw:initialize-common-windows)

(defclass strobe-camera (rigid-body)

  ((focal-length

    :accessor focal-length

    :initform 150)

   (posture

    :accessor posture   ; x y z phi theta psi

    :initform (list 0 0 0 0 0 0))

   (camera-window

    :accessor camera-window
   :initform (cw:make-window-stream :borders 5
                                    :left 300
                                       :bottom 300
                                          :width 400    ;1000
                                          :height 400   ;900
                                    :title "Graphical Display of UAV and Projectiles"
                                    :background-color white
                                    :foreground-color white
                                          :activate-p t))
  (H-matrix
   :initform (homogeneous-transform 0 0 0 0 0 0))
  (inverse-H-matrix
   :accessor inverse-H-matrix
   :initform (inverse-H (homogeneous-transform 0 0 0 0 0 0)))
  (enlargement-factor
   :accessor enlargement-factor
   :initform 100)))
(defun create-camera-1 ()
 (setf camera-1 (make-instance 'strobe-camera))
 (queue-mouse camera-1))
; *** Defined global color constants ***************************************
; To be used as the draw-color argument in take-picture and new-picture
; functionss (and also jack-picture, jack-video, jack-movie functions)
(defconstant *white* 0)
(defconstant *yellow* 1)
```

```
(defconstant *red* 2)

(defconstant *green* 3)

(defconstant *black* 4)

(defconstant *cyan* 5)

(defconstant *magenta* 6)

(defconstant *blue* 7)

; *** Draw picture functions ***********************************************

(defmethod take-picture ((camera strobe-camera) (body rigid-body) draw-color)

  (let ((camera-space-node-list (mapcar #'(lambda (node-location)

         (post-multiply (inverse-H-matrix camera) node-location))

                        (transformed-node-list body))))

    (dolist (polygon (polygon-list body))

      (clip-and-draw-polygon camera polygon camera-space-node-list draw-color))))

(defmethod erase-camera-window ((camera strobe-camera))

  (cw:clear (camera-window camera)))


(defmethod erase-block ((camera strobe-camera) (body rigid-body))

  (let ((center (perspective-transform camera

              (post-multiply (inverse-H-matrix camera)

              (first (transformed-node-list body))))))

     (cw:draw-filled-rectangle (camera-window camera)

                       (make-position :x (- (first center) 150)

                                  :y (- (second center) 150))

                       300 300 :color white)))
```

```
(defmethod new-picture ((camera strobe-camera) (body rigid-body) draw-color)
   (erase-camera-window camera)
   (take-picture camera body draw-color))
(defmethod clip-and-draw-polygon
  ((camera strobe-camera) polygon node-coord-list draw-color)
  (do* ((initial-point (nth (first polygon) node-coord-list))
        (from-point initial-point to-point)
        (remaining-nodes (rest polygon) (rest remaining-nodes))
        (to-point (nth (first remaining-nodes) node-coord-list)
                  (if (not (null (first remaining-nodes)))
                      (nth (first remaining-nodes) node-coord-list))))
       ((null to-point)
            (draw-clipped-projection camera from-point initial-point draw-color))
       (draw-clipped-projection camera from-point to-point draw-color)))
(defmethod draw-clipped-projection ((camera strobe-camera)
         from-point to-point draw-color)
  (cond ((and (<= (first from-point) (focal-length camera))
              (<= (first to-point) (focal-length camera))) nil)
        ((<= (first from-point) (focal-length camera))
         (draw-line-in-window camera
           (perspective-transform camera (from-clip camera from-point to-point))
           (perspective-transform camera to-point) draw-color))
        ((<= (first to-point) (focal-length camera))
         (draw-line-in-window camera
           (perspective-transform camera from-point)
        (perspective-transform camera (to-clip camera from-point to-point))
         draw-color))
      (t (draw-line-in-window camera
        (perspective-transform camera from-point)
        (perspective-transform camera to-point) draw-color))))


(defmethod from-clip ((camera strobe-camera) from-point to-point)
  (let ((scale-factor (/ (- (focal-length camera) (first from-point))
                        (- (first to-point) (first from-point)))))
     (list (+ (first from-point)
            (* scale-factor (- (first to-point) (first from-point))))
          (+ (second from-point)
            (* scale-factor (- (second to-point) (second from-point))))
          (+ (third from-point)
            (* scale-factor (- (third to-point) (third from-point)))) 1)))
(defmethod to-clip ((camera strobe-camera) from-point to-point)
  (from-clip camera to-point from-point))
(defmethod draw-line-in-window ((camera strobe-camera) start end draw-color)
   (cond ((= 0 draw-color) (cw:draw-line (camera-window camera)
            (cw:make-position :x (first start) :y (second start))
            (cw:make-position :x (first end) :y (second end))
            :brush-width 0 :color white))
      ((= 1 draw-color) (cw:draw-line (camera-window camera)
            (cw:make-position :x (first start) :y (second start))
            (cw:make-position :x (first end) :y (second end))
        :brush-width 0 :color yellow))
   ((= 2 draw-color) (cw:draw-line (camera-window camera)
        (cw:make-position :x (first start) :y (second start))
        (cw:make-position :x (first end) :y (second end))
        :brush-width 0 :color red))
   ((= 3 draw-color) (cw:draw-line (camera-window camera)
        (cw:make-position :x (first start) :y (second start))
        (cw:make-position :x (first end) :y (second end))
        :brush-width 0 :color green))
   ((= 4 draw-color) (cw:draw-line (camera-window camera)
        (cw:make-position :x (first start) :y (second start))
        (cw:make-position :x (first end) :y (second end))
        :brush-width 0 :color black))
   ((= 5 draw-color) (cw:draw-line (camera-window camera)
        (cw:make-position :x (first start) :y (second start))
        (cw:make-position :x (first end) :y (second end))
        :brush-width 0 :color cyan))
   ((= 6 draw-color) (cw:draw-line (camera-window camera)
        (cw:make-position :x (first start) :y (second start))
        (cw:make-position :x (first end) :y (second end))
        :brush-width 0 :color magenta))
   ((= 7 draw-color) (cw:draw-line (camera-window camera)
        (cw:make-position :x (first start) :y (second start))
        (cw:make-position :x (first end) :y (second end))
        :brush-width 0 :color blue))))
```

```
(defmethod perspective-transform ((camera strobe-camera) point-in-camera-space)
 (let* ((enlargement-factor (enlargement-factor camera))
        (focal-length (focal-length camera))
        (x (first point-in-camera-space))  ;x axis is along optical axis
        (y (second point-in-camera-space)) ;y is out right side of camera
        (z (third point-in-camera-space))) ;z is out bottom of camera
       (list (+ (round (* enlargement-factor (/ (* focal-length y) x)))
                    200)                    ;to right in camera window
           (+ 200 (round (* enlargement-factor (/ (* focal-length (- z)) x))
                         ))))) ;up in camera window ;500 500
; *** Position camera functions ****************************************
(defmethod move-camera ((camera strobe-camera) azimuth elevation roll x y z)
 (setf (posture camera) (list x y z roll elevation azimuth))
 (setf (H-matrix camera) (homogeneous-transform azimuth elevation roll x y z))
 (setf (inverse-H-matrix camera) (inverse-H (H-matrix camera))))
(defmethod zoom-camera ((camera strobe-camera) zoom-amount)
 (setf (slot-value camera 'enlargement-factor)
      (+ (slot-value camera 'enlargement-factor) zoom-amount)))
(defun deg-to-rad (angle) (* .017453292519943295 angle))
(defconstant tilt-limit (* .017453292519943295 89.9))
; *** Auxiliary functions ***********************************************
(defun kill ()
 (cw:kill-common-windows))
(defun reset-windows ()
(kill)

(cw:initialize-common-windows))
```

APPENDIX D: SOURCE CODE (PERFECT AUTOPILOT FILE)

```
; File: pioneer-perfect-autopilot.cl          Franz Common LISP

; ** PERFECT AUTOPILOT CLASS  DEFINITION **

; A Perfect Autopilot steers the body axes of the rigid-body to the

; desired orientation with no time delay.

; Requires: pioneer-euler-angle-rigid-body.cl

; Code written by R.B. McGhee, Naval Postgraduate School, & modified by D. Wiley

; mcghee@cs.nps.navy.mil

;*********************************************************************

(defclass perfect-autopilot ()

 ((vehicle-name

   ; This is the name of an instance of the rigid-body class.

   :accessor vehicle-name)

  (current-trajectory-segment

   :accessor current-trajectory-segment)

  (current-time

   :initform 0

   :accessor current-time)

  (longitudinal-acceleration-gain

   :accessor longitudinal-acceleration-gain)

  (trajectory-segment-list

   :accessor trajectory-segment-list)))

   ;This is a list of lists. Each list contains

   ;start-time and commanded speed,

   ;heading-rate, and depth. Last segment is end-time followed by nil.
```

```
(defmethod initialize-4 ((autopilot perfect-autopilot) vehicle gain trajectory)
  (setf (longitudinal-acceleration-gain autopilot) gain
        (trajectory-segment-list autopilot) (rest trajectory)
        (current-trajectory-segment autopilot) (first trajectory)
        (vehicle-name autopilot) vehicle))
(defmethod update-segment ((autopilot perfect-autopilot) time)
  (if (and (not (null (second (current-trajectory-segment autopilot))))
           (>= time (caar (trajectory-segment-list autopilot))))
      (setf (current-trajectory-segment autopilot)
            (pop (trajectory-segment-list autopilot)))))
(defmethod commanded-velocity ((autopilot perfect-autopilot) delta-t)
  (setf (current-time autopilot) (+ (current-time autopilot) delta-t))
  (update-segment autopilot (current-time autopilot))
  (if (second (current-trajectory-segment autopilot))
      (list (+ (first (velocity (vehicle-name autopilot)))
               (* (longitudinal-acceleration autopilot) delta-t))
            0 0 (fourth (current-trajectory-segment autopilot))
            (fifth (current-trajectory-segment autopilot))
            (third (current-trajectory-segment autopilot)))))
(defmethod longitudinal-acceleration ((autopilot perfect-autopilot))
  (* (longitudinal-acceleration-gain autopilot)
     (- (second (current-trajectory-segment autopilot))
        (first (velocity (vehicle-name autopilot))))))
```

```
(defmethod move-vehicle     ((autopilot perfect-autopilot) delta-t)
 (setf (velocity (vehicle-name autopilot))
    (commanded-velocity autopilot delta-t))
 (when (second (velocity (vehicle-name autopilot)))
  (update-posture (vehicle-name autopilot) delta-t)
  (setf (H-matrix (vehicle-name autopilot))
     (homogeneous-transform (sixth (posture (vehicle-name autopilot)))
                (fifth (posture (vehicle-name autopilot)))
                (fourth (posture (vehicle-name autopilot)))
                (first (posture (vehicle-name autopilot)))
                (second (posture (vehicle-name autopilot)))
                (third (posture (vehicle-name autopilot)))))
  (transform-node-list (vehicle-name autopilot))))
(defmethod accelerometer-output ((autopilot perfect-autopilot))
 (let ((longitudinal-velocity (first (velocity (vehicle-name autopilot))))
    (pitch-angle (fifth (posture (vehicle-name autopilot))))
    (roll-angle (fourth (posture (vehicle-name autopilot))))
    (pitch-rate (fifth (velocity (vehicle-name autopilot))))
    (yaw-rate (sixth (velocity (vehicle-name autopilot)))))
   (list (+ (longitudinal-acceleration autopilot)
       (* *gravity* (sin pitch-angle)))
     (- (* longitudinal-velocity yaw-rate)
       (* *gravity* (cos pitch-angle) (sin roll-angle)))
     (+ (- (* longitudinal-velocity pitch-rate))
       (- (* *gravity* (cos pitch-angle) (cos roll-angle)))))))
```

```
(defmethod mission-data ((autopilot perfect-autopilot))
  (append (IMU-data autopilot)
          (list (sixth (posture (vehicle-name autopilot))))))
(defmethod IMU-data ((autopilot perfect-autopilot))
  (cons (current-time autopilot)
        (append (accelerometer-output autopilot)
                (angular-rate-output autopilot))))
(defmethod angular-rate-output ((autopilot perfect-autopilot))
  (cons (fourth (velocity (vehicle-name autopilot)))
        (cons (fifth (velocity (vehicle-name autopilot)))
              (list (sixth (velocity (vehicle-name autopilot)))))))
(defun initialize-mission (*X* *Y* *Z* *AZM*)
  (setf airplane-1 (make-instance 'rigid-body))
  (setf autopilot-1 (make-instance 'perfect-autopilot))
  (initialize-4 autopilot-1 airplane-1 1 *trajectory*)
  (move-vehicle autopilot-1 0)
  (setf camera-1 (make-instance 'strobe-camera))
  (move-camera camera-1 0 0 0 0 0 0)
  (setf (first(posture airplane-1)) *X*)
  (setf (second(posture airplane-1)) *Y*)
  (setf (third(posture airplane-1)) *Z*)
  (setf (sixth(posture airplane-1)) *AZM*)
  (new-picture camera-1 airplane-1 *black*)
  (setf *bullets-fired* 0)
  (perfect-gunner-aim-shoot airplane-1 2)
  (dolist (bullet bullet-1) (take-picture camera-1 bullet *red*)))
  ;(queue-mouse))
(defun execute-mission ()
  (do* ((firecontrol 0 (+ firecontrol 1))
        (mission-data (list (mission-data autopilot-1))
                      (cons (mission-data autopilot-1) mission-data))
        (new-node-list (move-vehicle autopilot-1 .1)
                   (move-vehicle autopilot-1 .1)))
       ((not (second (velocity (vehicle-name autopilot-1))))
        (setf *mission-data* (reverse mission-data)))
        (update_bullets bullet-1)
        (dolist (bullet bullet-1)
             (if (close-to-hit (posture airplane-1)(posture bullet))
                 (test-for-hit bullet airplane-1)))
        (if (and (> (first (posture airplane-1)) 1)
              (< (second (posture airplane-1)) .1))
            (gunner-fire-bullet-in *azimuth* *elevation* firecontrol))
        (if (and (< (first (posture airplane-1)) 1)
              (< (second (posture airplane-1)) .1))
            (gunner-fire-bullet-out *azimuth* *elevation* firecontrol))
        (if (>(second (posture airplane-1)) .1) (gunner-fire-bullet-offset
                                        *azimuth* *elevation* firecontrol))
        (new-picture camera-1 airplane-1 *black*)
        (dolist (bullet bullet-1) (take-picture camera-1 bullet *red*))))
```

```
(setf *trajectory* '((0 60 0 0 0)  (100 nil)))
```

APPENDIX E: SOURCE CODE (DETERMINE A HIT FILE)

```
; File: determine-hit.cl                    Franz Common LISP

; ** Functions to test a projectile for penetration of a **

; ** UAV surface                                       **

; Code written by D. WILEY, Naval Postgraduate School, dawiley@cs.nps.navy.mil

; **********************************************************************

(defconstant *distance* 223)

(defconstant *range-of-projectile* 6200)

(defun test-for-hit (bullet plane)

  (let ((bullet-prev (convert-bullet (old-posture bullet)plane))

      (bullet-curr (convert-bullet (posture bullet) plane)))

    (cross-infinite-planep bullet-prev bullet-curr bullet)))

(defun convert-bullet (bullet plane)

  (let ((bullet-pos (bullet-in-plane-coordinates bullet plane)))

    (list (first bullet-pos)(second bullet-pos)(third bullet-pos))))

(defun bullet-in-plane-coordinates (bullet plane)

  (let ((xb (first bullet))(yb (second bullet))

       (zb (third bullet)))

  (post-multiply (inverse-H (H-matrix plane))(list xb yb zb 1))))

;Tests to see if the old bullet position and its current bullet have crossed

;the geometric plane that the UAV component lies in, if that has happened

;it then calls a function to find the

;X Y and Z coordinates of the bullet on the geometric plane

(defun cross-infinite-planep (bullet-prev bullet-curr bullet-ID)

  (dolist (plane *uav-components*)
 (let* ((uav-component plane)
        (point-on-geo-plane (first plane))
        (vec12 (vector-subtract (second plane)(first plane)))
        (vec14 (vector-subtract (fourth plane)(first plane)))
        (normal-plane (cross-product vec14 vec12))
        (vec1-bulletcurr(vector-subtract bullet-curr (first plane)))
        (vec1-bulletprev(vector-subtract bullet-prev (first plane)))
        (result1 (dot-product normal-plane vec1-bulletcurr))
        (result2 (dot-product normal-plane vec1-bulletprev)))
    (if (signs-samep result1 result2) t
         (bullet-on-plane bullet-prev bullet-curr normal-plane
                     point-on-geo-plane uav-component bullet-ID)))))
(defun signs-samep (x y)
  (if (or (and (>= x 0)(>= y 0))(and (<= x 0)(<= y 0))) t))
;Determines the X Y and Z coordinate of a bullet on a geometric plane
;given the previous position, new position and a point on the geometric plane.
(defun bullet-on-plane (bullet-prev bullet-curr normal point uav-component bullet-ID)
  (let* ((bpath (vector-subtract bullet-curr bullet-prev))
        (xpath (first bpath))(ypath (second bpath))(zpath (third bpath))
        (a (first normal))(b (second normal))(c (third normal))
        (x (first bullet-prev))(y (second bullet-prev))(z (third bullet-prev))
        (xPlane (first point))(yPlane (second point))(zPlane (third point))
        (d (- (+(* a xPlane)(* b yPlane)(* c zPlane))))
        (k (-(/(+(* a x)(* b y)(* c z)d)(+(* a xpath)(* b ypath)(* c zpath)))))
        (planeX (+ x (* k xpath)))
```

```
                    (planeY (+ y (* k ypath)))(planeZ (+ z (* k zpath))))
     (angle-sum-test (list planeX planeY planeZ) uav-component bullet-ID)))
;Tests a bullet to see if it lies om any part of a rectangle on a plane by
;summing the angle between vectors and checking for the sum to be close to 2 pi
(defun angle-sum-test (bullet-on-plane UAV-surface bullet-ID)
 (let* ((p UAV-surface)(b bullet-on-plane)
       (v1 (vector-subtract (first p) b))(v2 (vector-subtract (second p) b))
       (v3 (vector-subtract (third p) b))(v4 (vector-subtract (fourth p) b))
       (angle-sum (+ (angle-between-vectors v1 v2)(angle-between-vectors v2 v3)
                    (angle-between-vectors v3 v4)(angle-between-vectors v4 v1))))
    (if (and (> angle-sum 6.25)(< angle-sum 6.30))  ; threshold for around 2 pi
          (provide-feedback p bullet-ID))))
(defun angle-between-vectors (v1 v2)
  (let ((result (/ (dot-product v1 v2)
                   (*(vector-magnitude v1)(vector-magnitude v2)))))
    (if (>= result 1.0) (acos 1.0)(acos result))))
(defun close-to-hit (plane bullet)
  (let ((xp (first plane))  (yp (second plane)) (zp (third plane))
        (xb (first bullet)) (yb (second bullet)) (zb (third bullet)))
       (if (< (distance-bullet-plane xp xb yp yb zp zb) *distance*) t)))
(defun distance-bullet-plane (xp xb yp yb zp zb)
  (sqrt (+(square(- xp xb))(square(- yp yb))(square(- zp zb)))))
(defun update_bullets (bullet-list)
  (cond ((null bullet-list) nil)
        (t (update-rigid-body (first bullet-list))
        (if (relevent-bullet airplane-1 (first bullet-list))
            (setf bullet-1 (cons (first bullet-list)
                            (update_bullets (rest bullet-list))))))))
(defun relevent-bullet (plane bullet)
  (if (and (above-ground-bullet (third (posture bullet)))
        (< (distance-gun-object bullet) *range-of-projectile*))t))
 (defun above-ground-bullet (projectile)
  (if (< projectile 1.0) t))
(defun distance-gun-object (object)
  (let ((x (first (posture object)))(y (second (posture object)))
      (z (third (posture object))))
      (sqrt (+ (square x)(square y)(square z)))))
```

APPENDIX F: SOURCE CODE (ADJUST FIRE FILE)

```
; File: adjust-fire.cl          Franz Common LISP

;

; ** Gunners elevation and azimuth computations **

; A Perfect Gunner establishes a wall of fire in front

; of a UAV until the UAV is killed or past the wall

; of fire

; Code written by D. A. Wiley, Naval Postgraduate School,

; dawiley@cs.nps.navy.mil

;*****************************************************************

(defconstant *super-elevation* .0175) ; approximately 1 degree .0175

(defconstant *max-elevation-of-gun*  1.047197)

(defconstant *specified-velocity* 3608)

(defconstant *velocity-dispersion-feet* 16.4)   ;+- 5 meters per shot = 16.4 ft

(defconstant *elevation-step* .005)

(defconstant *lead-radians* .02)

(defconstant *angular-dispersion-radians*  0.00087375)

(defun gunner-fire-bullet-offset (azimuth elevation firecontrol)

  (if (or(= (length bullet-1) 0)

        (>(azimuth-angle (first bullet-1))(- (azimuth-angle airplane-1)

          *lead-radians*)))

        ; fire a bullet if there has not been one fired or

        ; your azimuth to bullet is > azimuth to plane

      (cond ((evenp firecontrol)

             (let ((x (first (posture camera-1)))
              (y (second (posture camera-1)))
              (z (third (posture camera-1)))
              (roll (fourth (posture camera-1)))
              (bullet (make-instance 'bullet))
              (azmRandom (get-random-angle azimuth))
              (elevRandom (get-random-angle elevation)))
              (initialize bullet)
              (move-body bullet azmRandom elevRandom roll x y z)
              (move-camera camera-1 azimuth elevation roll x y z)
              (setf (first(velocity bullet)) (get-random-velocity))
              (push bullet bullet-1)
              (setf *bullets-fired* (+  *bullets-fired* 1))
              (if (>= (fifth (posture(first bullet-1)))
                      *max-elevation-of-gun*)
                (pprint "At positive elevation limit")))
           t t))
     (perfect-gunner-aim-shoot airplane-1 2)))
(defun gunner-fire-bullet-in (azimuth elevation firecontrol)
 (if (or(= (length bullet-1) 0)
        (>(fifth (posture (first bullet-1))) (- (shot-1-theta-1 airplane-1)
          *lead-radians*)))
         ;fire a bullet if there has not been one fired or if
         ;you are still in front of the target, *lead-radians*
         ;assures plane has passed through wall of fire
     (cond ((evenp firecontrol)
        (let ((x (first (posture camera-1)))
              (y (second (posture camera-1)))
              (z (third (posture camera-1)))
              (roll (fourth (posture camera-1)))
              (bullet (make-instance 'bullet))
              (azmRandom (get-random-angle azimuth))
              (elevRandom (get-random-angle elevation)))
              (initialize bullet)
              (move-body bullet azmRandom elevRandom roll x y z)
              (move-camera camera-1 azimuth elevation roll x y z)
              (setf (first(velocity bullet)) (get-random-velocity))
              (push bullet bullet-1)
              (setf *bullets-fired* (+  *bullets-fired* 1))
              (if (>= (fifth (posture(first bullet-1)))
                      *max-elevation-of-gun*)
                  (pprint "At positive elevation limit")))
        t t))
    (perfect-gunner-aim-shoot airplane-1 2)))
(defun gunner-fire-bullet-out (azimuth elevation firecontrol)
  (if (or (= (length bullet-1) 0)
        (< (fifth (posture (first bullet-1)))(+ (shot-1-theta-1 airplane-1)
           *lead-radians*)))
        ;fire a bullet if there has not been one fired or if
        ;you are still in front of the target, *lead-radians*
        ; assures plane has passed through wall of fire
    (cond ((evenp firecontrol)
          (let ((x (first (posture camera-1)))
               (y (second (posture camera-1)))
               (z (third (posture camera-1)))
               (roll (fourth (posture camera-1)))
               (bullet (make-instance 'bullet))
               (azmRandom (get-random-angle azimuth))
               (elevRandom (get-random-angle elevation)))
               (initialize bullet)
               (move-body bullet azmRandom elevRandom roll x y z)
               (move-camera camera-1 azimuth elevation roll x y z)
               (setf (first(velocity bullet)) (get-random-velocity))
               (push bullet bullet-1)
               (setf *bullets-fired* (+  *bullets-fired* 1))
               (if (>= (fifth (posture(first bullet-1)))
                     *max-elevation-of-gun*)
                  (pprint "At positive elevation limit")))
          t t))
     (perfect-gunner-aim-shoot airplane-1 2)))
(defun perfect-gunner-aim-shoot (UAV firecontrol)
  (format t "~%Bullets fired: ~A" *bullets-fired*)
  (format t "~%Location of UAV: ~A" (posture airplane-1))
  (format t "~%Range Gun to Target: ~A" (distance-gun-object airplane-1))
  (format t "~%Azimuth Gun to Target: ~A" (rad-to-deg (azimuth-angle UAV)))
  (setf *bullets-fired* 0)
```

```
 (setf bullet-1 nil)
 (if (and (> (first (posture UAV)) 1)(< (second (posture UAV)) .1))
    (determine-angles-inbound UAV firecontrol))
 (if (and (< (first (posture UAV)) 1)(< (second (posture UAV)) .1))
    (determine-angles-outbound UAV firecontrol))
 (if (>(second (posture UAV)) .1)
    (determine-angles-offset UAV firecontrol)))
(defun determine-angles-offset (UAV firecontrol)
 (let* ((theta-1 (shot-1-theta-1 UAV))
        (theta-2 (shot-1-theta-2-inbound UAV theta-1))
        (theta-3 (shot-1-theta-3-inbound UAV))
        (azm    (azimuth-angle (UAV-at-future-point UAV))))
    (setf *azimuth* azm)
    (format t  "~%~%Computing angles for elevation/azimuth to lay wall of fire!")
    (setf *elevation* (determine-elevation-offset UAV))
    (gunner-fire-bullet-offset *azimuth* *elevation* firecontrol)))
(defun determine-elevation-offset (UAV)
(let ((new-UAV (UAV-at-future-point UAV)))
   (do ((elevation (get-elev-angle-to-UAV new-UAV)(+ elevation *elevation-step*))
        (close-flag nil))
        (close-flag elevation)
     (do ((test-bullet (make-test-bullet elevation)))
        ((or(not(relevent-bullet new-UAV test-bullet))
            close-flag
           (< (third (posture test-bullet))(third (posture new-UAV))))
        nil)
      (update-rigid-body test-bullet)
      (if (bullet-path-thru-UAV-plane-p new-UAV test-bullet)
            (setf close-flag 1))))))
(defun bullet-path-thru-UAV-plane-p (UAV bullet)
  (let ((uav-Z (third (posture UAV)))
       (prev-bullet-Z (third (old-posture bullet)))
       (curr-bullet-Z (third (posture bullet))))
    (if (and (> prev-bullet-Z uav-Z)(< curr-bullet-Z uav-Z))t)))
(defun make-test-bullet (elevation)
  (let ((test-bullet (make-instance 'bullet)))
       (initialize test-bullet)
       (move-body test-bullet *azimuth* elevation 0 0 0 0)
       (setf (first (velocity test-bullet)) *specified-velocity*)
       test-bullet))
(defun UAV-at-future-point (UAV)
  (let((new-UAV (make-instance 'rigid-body)))
     (setf (first(posture new-UAV))
          (- (first (posture UAV))100))
     (setf (second(posture new-UAV)) (second(posture UAV)))
     (setf (third(posture new-UAV))(third(posture UAV)))
     (setf (sixth(posture new-UAV)) (sixth(posture UAV)))
    new-UAV))
(defun determine-angles-inbound (UAV firecontrol)
  (let* ((theta-1 (shot-1-theta-1 UAV))
        (theta-2 (shot-1-theta-2-inbound UAV theta-1))
        (theta-3 (shot-1-theta-3-inbound UAV))
        (azm     (azimuth-angle UAV)))
    (setf *azimuth* azm)
    (format t  "~%~%Computing angles for elevation/azimuth to lay wall of fire!")
    (setf *elevation* (+ (+ theta-3 (- theta-1 theta-2))*super-elevation*))
    (gunner-fire-bullet-in *azimuth* *elevation* firecontrol)))
(defun determine-angles-outbound (UAV firecontrol)
  (let* ((theta-1 (shot-1-theta-1 UAV))
        (theta-2 (shot-1-theta-2-outbound UAV theta-1))
        (theta-3 (shot-1-theta-3-outbound UAV))
        (azm     (azimuth-angle UAV)))
    (setf *azimuth* azm)
    (format t "~%Computing angles for elevation/azimuth to lay wall of fire!")
    (setf *elevation* (- (+ theta-3 (- theta-1 theta-2)) *super-elevation*))
    (gunner-fire-bullet-out *azimuth* *elevation* firecontrol)))
(defun shot-1-theta-1 (UAV)
   (let ((X (first (posture UAV)))
        (Y (second (posture UAV)))
        (Z (third (posture UAV))))
    (elev-angle X Y Z)))
(defun shot-1-theta-2-inbound (UAV theta-1)
  (let* ((Bullet (actual-shots-location-inbound theta-1 UAV))
       (X (first (posture Bullet)))
       (Y (second (posture UAV)))
```

```
      (Z (third (posture Bullet))))
  (elev-angle X Y Z)))
(defun shot-1-theta-2-outbound (UAV theta-1)
 (let* ((Bullet (actual-shots-location-outbound theta-1 UAV))
      (X (first (posture Bullet)))
      (Y (second (posture UAV)))
      (Z (third (posture Bullet))))
  (elev-angle X Y Z)))
(defun actual-shots-location-inbound (elevation-angle UAV)
  (setf *test-bullet* (make-instance 'bullet))
  (initialize *test-bullet*)
  (move-body *test-bullet* (azimuth-angle UAV) elevation-angle 0 0 0 0)
  (fire-test-bullet UAV))
(defun actual-shots-location-outbound (elevation-angle UAV)
  (setf *test-bullet* (make-instance 'bullet))
  (initialize *test-bullet*)
  (move-body *test-bullet* (azimuth-angle UAV) elevation-angle 0 0 0 0)
  (fire-test-bullet UAV))
(defun fire-test-bullet (UAV)
  (setf (time-of-flight *test-bullet*) 0)
  (setf (first (velocity *test-bullet*)) *specified-velocity*)
  (do ()
    ((and (> (abs (first (posture *test-bullet*)))
            (abs (first (posture UAV))))
       (<  (third(posture *test-bullet*)) (third(posture airplane-1)))))
    (update-rigid-body *test-bullet*)
    (update-time-of-flight *test-bullet*))
   *test-bullet*)
(defun shot-1-theta-3-inbound (UAV)
  (let ((X (- (first (posture UAV)) (* (* (time-of-flight *test-bullet*) .1) 6)))
        (Y (second (posture UAV)))
        (Z (third (posture UAV))))
   (elev-angle X Y Z)))
(defun shot-1-theta-3-outbound (UAV)
  (let ((X (+ (first (posture UAV)) (* (* (time-of-flight *test-bullet*) .1) -6)))
        (Y (second (posture UAV)))
        (Z (third (posture UAV))))
   (elev-angle X Y Z)))
(defun get-elev-angle-to-UAV (UAV)
  (let ((X (first (posture UAV)))
        (Y (second (posture UAV)))
        (Z (third (posture UAV))))
    (elev-angle X Y Z)))
(defun elev-angle (X Y Z)
  (atan (abs Z) (sqrt(+ (* X X)(* Y Y)))))
(defun azimuth-angle (UAV)
  (let ((X (first (posture UAV)))
        (Y (second (posture UAV))))
    (atan Y X)))
(defun update-time-of-flight (bullet)
 (setf (time-of-flight bullet) (+ (time-of-flight bullet)1)))
(defun update-time-of-flight (bullet)
 (setf (time-of-flight bullet) (+ (time-of-flight bullet)1)))
(defun get-random-velocity ()
  (+ *specified-velocity*  (* *velocity-dispersion-feet* (- (random 2.0) 1))))
(defun get-random-angle (angle)
  (+ angle (* *angular-dispersion-radians* (- (random 2.0) 1))))
```

APPENDIX G: SOURCE CODE (MOUSE HANDLER FILE)

```
; File: mouse.cl                    Franz Common LISP

; Mouse handler to allow adjusting azimuth/elevation of the launcher, zooming

; of the camera and firing of a bullet

; Must be invoked with the call (queue-mouse) at start of simulation

; Code written by D. T. Davis, Naval Postgraduate School, & modified by D. Wiley;

;*****************************************************************

(defconstant *1degree* 0.017453)

(defconstant *positive-elevation* 1.05)

(defconstant *negative-elevation* -0.158)

(defmethod queue-mouse ()

  (cw:sun-enable-super-and-hyper)

  (cw:modify-window-stream-method (camera-window camera-1) :left-button-down

                                  :after 'mouse-handler)

  (cw:modify-window-stream-method (camera-window camera-1) :middle-button-down

                                  :after 'mouse-handler)

  (cw:modify-window-stream-method (camera-window camera-1) :right-button-down

                                  :after 'mouse-handler))

(defun mouse-handler (wstream cw:mouse-state &optional event)

  (let ((button-state (cw:mouse-button-state)))

    ;(format t "Mouse Handler Invoked:  ~a ~%" button-state)

    (case button-state
        (128 (slew-left))    ; left click
        (64 (fire-bullet))   ; middle click
        (32 (slew-right))    ; right click
        (136 (zoom-out-X2))  ; left click + alt
        (40 (zoom-in-X2))    ; right-click + alt
        (144 (slew-up))      ; left click + shift
        (48 (slew-down))))) ; right click + shift
(defun slew-left ()
  (let ((x (first (posture camera-1)))
        (y (second (posture camera-1)))
        (z (third (posture camera-1)))
        (azimuth (- (sixth (posture camera-1)) *1degree*))
        (elevation (fifth (posture camera-1)))
        (roll (fourth (posture camera-1))))
       (move-camera camera-1 azimuth elevation roll x y z)))
(defun slew-right ()
  (let ((x (first (posture camera-1)))
        (y (second (posture camera-1)))
        (z (third (posture camera-1)))
        (azimuth (+ (sixth (posture camera-1)) *1degree*))
        (elevation (fifth (posture camera-1)))
        (roll (fourth (posture camera-1))))
```

```
    (move-camera camera-1 azimuth elevation roll x y z)))

(defun slew-up ()

 (cond ((< (fifth (posture camera-1)) *positive-elevation*)

        (let ((x (first (posture camera-1)))

              (y (second (posture camera-1)))

              (z (third (posture camera-1)))

              (azimuth (sixth (posture camera-1)))

              (elevation (+ (fifth (posture camera-1)) *1degree*))

              (roll (fourth (posture camera-1))))

             (move-camera camera-1 azimuth elevation roll x y z)))

        (t (pprint "At positive elevation limit"))))

(defun slew-down ()

 (cond ((> (fifth (posture camera-1)) *negative-elevation*)

        (let ((x (first (posture camera-1)))

              (y (second (posture camera-1)))

              (z (third (posture camera-1)))

              (azimuth (sixth (posture camera-1)))

              (elevation (- (fifth (posture camera-1)) *1degree*))

              (roll (fourth (posture camera-1))))

             (move-camera camera-1 azimuth elevation roll x y z)))

        (t (pprint "At negative elevation limit"))))

(defun zoom-in-X2 ()
  (let ((x (focal-length camera-1)))
      (if (<= x 1)
      (setf (focal-length camera-1) 2)
      (setf (focal-length camera-1) (* x 2))))
  (format t "Scope set to power: ~a ~%" (focal-length camera-1)))
(defun zoom-out-X2 ()
  (let ((x (focal-length camera-1)))
      (if (<= x 2)
          (setf (focal-length camera-1) 1)
          (setf (focal-length camera-1) (/ x 2))))
  (format t "Scope set to power: ~a ~%" (focal-length camera-1)))
(defun fire-bullet ()
  (let ((x (first (posture camera-1)))
      (y (second (posture camera-1)))  (z (third (posture camera-1)))
      (azimuth (sixth (posture camera-1)))
      (elevation (fifth (posture camera-1)))
      (roll (fourth (posture camera-1)))
      (bullet (make-instance 'bullet)))
      (initialize bullet)
     (move-body bullet azimuth elevation roll x y z)
     (setf (velocity bullet) '(3608 0 0 0 0 0))
     (push bullet bullet-1)))
```

APPENDIX H: SOURCE CODE (ROBOT KINEMATICS FILE)

```
; File: robot-kinematics.cl              Franz Common LISP

;

; Utility functions

; Code written by R.B. McGhee, Naval Postgraduate School, & modified by D. Wiley

; mcghee@cs.nps.navy.mil

; *****************************************************************

(defun transpose (matrix)        ;A matrix is a list of row vectors.

  (cond ((null (cdr matrix)) (mapcar 'list (car matrix)))

        (t (mapcar 'cons (car matrix) (transpose (cdr matrix))))))

(defun dot-product (vector-1 vector-2)  ;A vector is a list of numerical atoms.

  (apply '+ (mapcar '* vector-1 vector-2)))

(defun cross-product (v1 v2)

  (cons (- (* (second v1)(third v2))(* (second v2)(third v1)))

            (cons(- (* (first v2)(third v1))(* (first v1)(third v2)))

                 (list

                  (- (* (first v1)(second v2))(* (first v2)(second v1)))))))

(defun vector-magnitude (vector) (sqrt (dot-product vector vector)))

(defun square (x) (* x x))

(defun post-multiply (matrix vector)

  (cond ((null (rest matrix)) (list (dot-product (first matrix) vector)))

        (t (cons (dot-product (first matrix) vector)

                 (post-multiply (rest matrix) vector)))))

(defun pre-multiply (vector matrix)

  (post-multiply (transpose matrix) vector))
```

```
(defun matrix-multiply (A B)      ;A and B are conformable matrices.
  (cond ((null (cdr A)) (list (pre-multiply (car A) B)))
        (t (cons (pre-multiply (car A) B) (matrix-multiply (cdr A) B)))))
(defun chain-multiply (L)        ;L is a list of names of conformable matrices.
  (cond ((null (cddr L)) (matrix-multiply (eval (car L)) (eval (cadr L))))
        (t (matrix-multiply (eval (car L)) (chain-multiply (cdr L))))))
(defun cycle-left (matrix) (mapcar 'row-cycle-left matrix))
(defun row-cycle-left (row) (append (cdr row) (list (car row))))
(defun cycle-up (matrix) (append (cdr matrix) (list (car matrix))))
(defun unit-vector (one-column length)        ;Column count starts at 1.
  (do ((n length (1- n))
       (vector nil (cons (cond ((= one-column n) 1) (t 0)) vector)))
      ((zerop n) vector)))


(defun unit-matrix (size)
  (do ((row-number size (1- row-number))
       (I nil (cons (unit-vector row-number size) I)))
      ((zerop row-number) I)))
(defun concat-matrix (A B)   ;A and B are matrices with equal number of rows.
  (cond ((null A) B)
        (t (cons (append (car A) (car B)) (concat-matrix (cdr A) (cdr B))))))
(defun augment (matrix)
  (concat-matrix matrix (unit-matrix (length matrix))))
(defun normalize-row (row) (scalar-multiply (/ 1.0 (car row)) row))
(defun scalar-multiply (scalar vector)
 (cond ((null vector) nil)
       (t (cons (* scalar (car vector))
                (scalar-multiply scalar (cdr vector))))))
(defun solve-first-column (matrix)       ;Reduces first column to (1 0 ... 0).
 (do* ((remaining-row-list matrix (rest remaining-row-list))
       (first-row (normalize-row (first matrix)))
       (answer (list first-row)
               (cons (vector-add (first remaining-row-list)
                                 (scalar-multiply (- (caar remaining-row-list))
                                                  first-row))
                     answer)))
      ((null (rest remaining-row-list)) (reverse answer))))
 (defun vector-add (vector-1 vector-2) (mapcar '+ vector-1 vector-2))
(defun vector-subtract (vector-1 vector-2) (mapcar '- vector-1 vector-2))

(defun first-square (matrix)  ;Returns leftmost square matrix from argument.
  (do ((size (length matrix))
       (remainder matrix (rest remainder))
       (answer nil (cons (firstn size (first remainder)) answer)))
      ((null remainder) (reverse answer))))
 (defun firstn (n list)
  (cond ((zerop n) nil)
        (t (cons (first list) (firstn (1- n) (rest list))))))
 (defun max-car-firstn (n list)
  (append (max-car-first (firstn n list)) (nthcdr n list)))
```

```
(defun matrix-inverse (M)
 (do ((M1 (max-car-first (augment M))
        (cond ((null M1) nil)       ;Abort for singular matrix.
              (t (max-car-firstn n (cycle-left (cycle-up M1))))))
      (n (1- (length M)) (1- n)))
     ((or (minusp n) (null M1)) (cond ((null M1) nil) (t (first-square M1))))
     (setq M1 (cond ((zerop (caar M1)) nil) (t (solve-first-column M1))))))
(defun max-car-first (L)   ;L is a list of lists. This function finds list with
 (cond ((null (cdr L)) L) ;largest car and moves it to head of list of lists.
       (t (if (> (abs (caar L)) (abs (caar (max-car-first (cdr L))))) L
             (append (max-car-first (cdr L)) (list (car L)))))))
(defun dh-matrix (cosrotate sinrotate costwist sintwist length translate)
 (list (list cosrotate (- (* costwist sinrotate))
             (* sintwist sinrotate) (* length cosrotate))
       (list sinrotate (* costwist cosrotate)
             (- (* sintwist cosrotate)) (* length sinrotate))
       (list 0. sintwist costwist translate) (list 0. 0. 0. 1.)))
(defun homogeneous-transform (azimuth elevation roll x y z)
 (let ((spsi (sin azimuth)) (cpsi (cos azimuth)) (sth (sin elevation))
       (cth (cos elevation)) (sphi (sin roll)) (cphi (cos roll)))
      (list (list (* cpsi cth) (- (* cpsi sth sphi) (* spsi cphi))
                  (+ (* cpsi sth cphi) (* spsi sphi)) x)
            (list (* spsi cth) (+ (* cpsi cphi) (* spsi sth sphi))
                  (- (* spsi sth cphi) (* cpsi sphi)) y)
            (list (- sth) (* cth sphi) (* cth cphi) z)
```

```
          (list 0. 0. 0. 1.))))

(defun inverse-H (H)        ;H is a 4x4 homogeneous transformation matrix.
  (let* ((minus-P (list (- (fourth (first H)))
                        (- (fourth (second H)))
                        (- (fourth (third H)))))
       (inverse-R (transpose (first-square (reverse (rest (reverse H))))))
       (inverse-P (post-multiply inverse-R minus-P)))
      (append (concat-matrix inverse-R (transpose (list inverse-P)))
              (list (list 0 0 0 1)))))

(defun rotation-matrix (azimuth elevation roll)
  (let ((spsi (sin azimuth)) (cpsi (cos azimuth)) (sth (sin elevation))
      (cth (cos elevation)) (sphi (sin roll)) (cphi (cos roll)))
     (list (list (* cpsi cth) (- (* cpsi sth sphi) (* spsi cphi))
                 (+ (* cpsi sth cphi) (* spsi sphi)))
           (list (* spsi cth) (+ (* cpsi cphi) (* spsi sth sphi))
                 (- (* spsi sth cphi) (* cpsi sphi)))
           (list (- sth) (* cth sphi) (* cth cphi)))))


(defun body-rate-to-euler-rate-matrix (azimuth elevation roll)
  (let ((sth (sin elevation)) (cth (cos elevation)) (tth (tan elevation))
      (sphi (sin roll)) (cphi (cos roll)))
     (list (list 1 (* tth sphi) (* tth cphi))
           (list 0 cphi (- sphi))
           (list 0 (/ sphi cth) (/ cphi cth)))))
```

```
(defun rad-to-deg (angle) (* 57.2957795130823 angle))

(defun deg-to-rad (angle) (* .0174532925199433295 angle))
```

APPENDIX I: SOURCE CODE (COMPILE AND LOAD FILES)

```
; File: compile-files.cl          Franz Common LISP

; Defines basic function to run simulation and compiles all

; files used in the simulation.

; To run simulation, inside the CLOS interpreter do the following:

; > (load "compile-files")

; > (load "load-files")

; > (test X Y Z)        ; where X Y Z are the coordinates of the UAV

;                       ; Z is negative for altitudes above ground.

; Code written by D. A. Wiley, Naval Postgraduate School,

; dawiley@cs.nps.navy.mil

; *****************************************************************

(compile-file "strobe-camera.cl")

(compile-file "robot-kinematics.cl")

(compile-file "pioneer-euler-angle-rigid-body.cl")

(compile-file "pioneer-perfect-autopilot.cl")

(compile-file "bullet.cl")

(compile-file "mouse.cl")

(compile-file "determine-hit.cl")

(compile-file "uav-components.cl")

(compile-file "adjust-fire.cl")


;UAV will fly from north to south along X axis

(defun test (X Y Z)

  (initialize-mission X Y Z  3.141592653589793d0)
```

```
 (execute-mission))
;**************
; File: Load-files.cl              Franz Common LISP
;
; Loads compiled files
; Code written by D. A. Wiley, Naval Postgraduate School,
; dawiley@cs.nps.navy.mil
;*********************************************************************
(load "strobe-camera.fasl")
(load "robot-kinematics.fasl")
(load "pioneer-euler-angle-rigid-body.fasl")
(load "pioneer-perfect-autopilot.fasl")
(load "bullet.fasl")
(load "mouse.fasl")
(load "determine-hit.fasl")
(load "uav-components.fasl")
(load "adjust-fire.fasl")
```

LIST OF REFERENCES

1. Brainin, S. and McGhee, R., "*Optimal Biased Proportional Navigation,*" IEEE Transactions, v. AC-13, No. 4, 1968, pp. 440-442.

2. Cooke, J., Zyda, M., Pratt, D., and McGhee, R., "*NPSNET: Flight Simulation Dynamic Modeling Using Quaternions,*" Presence, v. 1, No. 4, 1992, pp. 404-420.

3. Craig, J., *Introduction to Robotics: Mechanics and Control*, Second Edition, Addison-Wesley Publishing Company, Inc., Menlo Park, California, 1989.

4. Davidson, S., *An Experimental Comparison of CLOS and C++ Implementations of an Object-Oriented Graphical Simulation of Walking Robot Kinematics*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1993.

5. Graham, P., "*ANSI Common Lisp,*" Prentice Hall, Englewood Cliffs, New Jersey, 1996.

6. Green, G., "*Air Force Activates "First-Ever" UAV Unit,*", Unmanned Systems, v. 13, No. 3, 1995, pp.42-45.

7. Hooton, E. and Munson K., "*Jane's Battlefield Surveillance Systems,*" Sixth Edition 1994-95, pp. 203-206.

8. Koschmann, T., *The Common LISP Companion*, John Wiley & Sons, United States of America, 1990.

9. Schultz, A., *From Indian Scouts To Robot Spy Planes,* August Issue, Arizona Flyways.

10. Steele, G., *Common LISP*, Digital Equipment Corporation, United States of America, 1990.

11. Stein, J., ed., *The Random House College Dictionary*, Revised Edition, Random House, Inc., New York, New York, 1979.

12. Siegel, B., "*Navy Prepared to Aid Operation Joint Endeavor with Unmanned Aircraft,*" Navy News Service: Bosnia Operations, - Navy Wire Service "A" NWSA1409., USS Shreveport Public Affairs, 4 JAN 1996

13. Street, B., *UAV Support for FA Operations*, Field Artillery, April 95 issue, pp. 34-36.

14. Hooton, E. and Munson K., "*Jane's Armored Personnel Carriers,*" Sixth Edition 1994-95, pp. 436-438.

15. US Army Writers., "STINGER Fighting Vehicle," Draft Edition, 1995.

16. US Army Writers., "Fighting Vehicle, Infantry," Current Edition, 1986 with changes 1,2,3,4 and 5.

17. Bundy, M., Schnell B.,and Maribelle, F., 1996, Army research Laboratory, Telephone conversation 24 January 1996, email 30 April 1996.

18. Williams, A., 1996, Pioneer UAV Detachment, Telephone conversation 3 February 1996.

19. Field Manual (FM) 44-3, Air Defense Operational Requirements Document (ORD) for FAAD C3I, 12 JUNE 95. TRW Inc., Space & Defense Sector, 9 August 1995.

20. US Army Writers., "Field Manual 44-43," Current Edition, 1994.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
Naval Postgraduate School
411 Dyer Rd.
Monterey, CA 93943-5101

3. Chairman, Code CS 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

4. Dr. Robert B. McGhee, Code CS/Mz 2
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

5. Major Leroy Jackson 2
TRADOC Analysis Center
PO Box 8692
Monterey, CA 93943

6. Major Earnest D. Harris 2
U.S. Army Artificial Intelligence Center
107 Army Pentagon
Washington, DC 20310-0107

7. Commander, USAADASCH.................................... 1
Attn: ATSA CDM F (MAJ Miller)
Ft. Bliss, TX 79916

8. LTC Anthony G. Wiley.. 2
Defense Systems Management College
12997 Hampton Forest Court
Fairfax, VA 22030

9. CPT Michael J. Sherrill...1
APMS University Central Arkansas ROTC
5 Sweetbriar Lane
Greenbrier, AR 72058

10. Kevin Crosthwaite.. 1
WL/FIVS/SURVIAC BLDG 45
2130 Eighth St, Ste 1
Wright Patterson AFB,Ohio
45433-7542

11. CPT Danny A. Wiley 2
HQ USEUCOM
Unit 30400 Box 1019
APO AE 09128